第３章　分散並列型動的ソルバー

3.1 はじめに

本章では、どのように PC クラスターを構築するのか、動的解析の中で、どのタイミングでマスターとスレーブ間における情報の送受信が必要となるのか、また、その情報とはどのような種類なのか、どのように送受信を行うのか、これらの課題について解説する。

分散並列型動的解析システムは、SPACE の動的解析システムを元に構築されており、解析手法や解析モデル、部材モデル、履歴モデルなどは、SPACE の仕様と全て同一となっている。それらについての説明は、SPACE のマニュアル：動的解析編を参照されたい。前章では、並列解析手法の説明と数値解析の流れ、及び、どのような情報がマスターとスレーブ間で送受信する必要があるかについて概観した。ここでは、具体的にプログラムコードを用いて分散並列型システムの構築と数値解析時におけるデータの送受信について説明する。

3.2 分散並列型動的解析システム

3.2.1 分散並列型システム構築の概要

本節では、分散並列型システム（PC クラスター）をどのように構築しているかについて説明する。分散並列型システムの構築は、MPI の仕様に沿って行っており、具体的にはプロセス設定ファイルを用いて行う。このファイルの仕様、並びに入力・管理については次節で述べる。

最初に分散並列処理システムの構築について、実際のプログラム実行過程にしたがって具体的に説明しよう。分散並列型動的解析システムが SPACE から起動されると、まず、クラス MainFrame のコンストラクターが実行され、グラフィックシステムで必要となる動的領域の確保と構造データの入力が行われる。これらに関する詳細は、マニュアル：動的解析編を参照されたい。次に、親ウインドウと子ウインドウが、画面上に各ひとつ表示される。

分散並列型動的解析システムは、SPACE のメニューから起動される。
動的解析は、メニューもしくは解析スタートツールチップを押すことで開始する。

子ウインドウの管理は、クラス Sf3stView で行っており、同様に動的ソルバーも同じクラスの中で管理されている。現在のバージョンでは、分散並列システムの構築は、解析の開始直後に行われる。これから説明するコードは全てクラス Sf3stView に記述されている。

動的解析の開始は、メニューの解析開始か、もしくは動的解析開始ツールチップを押すことで始まる。解析が開始されると、クラス Sf3stView 内のメンバー関数 OnAnalysisStart()に制御が移り、その関数の中で、次のコードによって分散並列システムが構築される。

```
CDlgSetProcess      dlg;
if(dlg.DoModal() == IDCANCEL)    return;
// 並列処理の初期化
if(!cMPI_Init(ID_MASTER, (LPCTSTR)dlg.m_strFile))
{
        MessageBox("初期化に失敗しました¥n");
        return;
}
```

分散並列システムを構築するために必要となる情報は、MPI の仕様では、プロセス設定ファイルから取得することになっている。そこで、ファイルを作成するために、ダイアログクラスである CdlgSetProcess のオブジェクト dlg を作り、次に、図 3-1 に示すモーダルダイアログを表示させ、ダイアログの中でデータを設定した後、並列構築用のプロセス設定ファイルを作成する。この処理については、第 3.4 節で解説する。次に、MPI_Init()の拡張版である初期化関数 cMPI_Init()をコールし、この関数の中で分散並列型システムが構築される。この分散並列型システムの構築については、第 3.3 節で詳細に述べることになる。この構築に成功すると次のステップに進むが、失敗するとエラーメッセージを表示した後、この関数から抜けることになる。

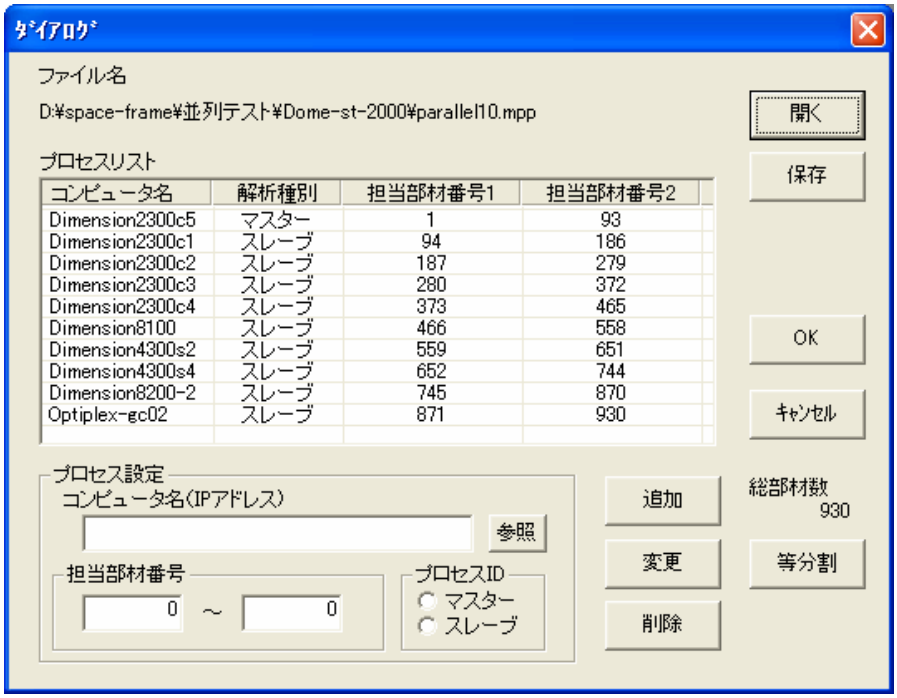


図 3-1 並列システム構築用ダイアログ

並列システム構築に成功すると、次のコードで示されるように関数 SUBMAIN_DYNAMIC_A()がコールされ、予備計算が実行される。この処理が終了すると図形処理を行う関数 Ontime()が実行され、さらに、関数 OnAnalysisGo()によって、実際の動的解析が開始され、以後は、マルチスレッド処理となる。この部分に関する処理の詳細はマニュアル：動的解析編を参照されたい。以下には、上記の処理を実行するクラス Sf3stView 内のメンバー関数 OnAnalysisStart()のコードを示す。これで、分散並列システムの構築と予備計算までのマスター側の処理について、その概要を示したことになる。

```
// 予備計算 FORTRAN サブルーチンを呼ぶ
      SUBMAIN_DYNAMIC_A()
//    ―――――――――――――――――――――――――――――
//      ●　図形処理
//    ―――――――――――――――――――――――――――――
      Ontime();
     MessageBox("予備計算終了:計算開始");
//    ―――――――――――――――――――――――――――――
//      ●　解析開始：マルチスレッド処理開始
//    ―――――――――――――――――――――――――――――
      OnAnalysisGo() ;
```

3.2.2 分散並列型動的解析

前節では、分散並列システムを構築する一連の処理について、流れに従って説明した。本節では、次のような並列処理に関連する内容を、さらに詳細に検討する。

１．PC クラスターを構築するためのファイル設定
２．具体的に PC クラスターの構築
３．各種のデータ転送
４．マルチスレッド処理による動的解析の開始

最初に、クラスSf3stView内で、並列処理に関連するコードを抜き出し、以下に示す。ここで説明するクラス Sf3stView のメンバー関数は、以下の５つである。

```
UINT threadprocx(LPVOID pParam)
void CSf3stView::OnAnalysisGo()
void CSf3stView::OnAnalysisStart()
LONG CSf3stView::OnMessageWind(UINT wParam, LONG lParam)
void CSf3stView::OnDynClear()
```

これらの処理がどのように行われているかを、具体的なコードから理解されたい。

```
//    ―――――――――――――――――――――――――――――――――――
//      ●　CSf3stView クラス
//    ―――――――――――――――――――――――――――――――――――
//      ●　CSf3stView クラスの動作の定義を行います
//    ―――――――――――――――――――――――――――――――――――
//      ●　ヘッダーファイルの定義
```

```
//    ――――――――――――――――――――――――――――――――――――――――――――――――
#include "stdafx.h"
#include "sf3st.h"
#include "sf3stdat.h"
#include "fort.h"
#include "DlgSetProcess.h"
#include "sf3stDoc.h"
#include "sf3stView.h"
#include "sf3stView.h"
#include "DialogView.h"
#include "MainFrm.h"
#include <fstream.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//    ――――――――――――――――――――――――――――――――――――――――――――――――
//      ●　メッセージによる関数呼び出し
//    ――――――――――――――――――――――――――――――――――――――――――――――――
IMPLEMENT_DYNCREATE(CSf3stView, CView)
BEGIN_MESSAGE_MAP(CSf3stView, CView)                                    // 1
      //{{AFX_MSG_MAP(CSf3stView)
      ON_WM_RBUTTONDOWN()
      ON_WM_LBUTTONDOWN()
      ON_WM_LBUTTONUP()
      ON_WM_MOUSEMOVE()
      ON_COMMAND(ID_SWND_ST, OnSwndSt)
      ON_COMMAND(ID_WND_PROP, OnWndProp)
      ON_COMMAND(ID_DYN_MAG, OnDynMag)
      ON_COMMAND(ID_SWND_WV_Y, OnSwndWvY)
      ON_COMMAND(ID_SWND_WV_Z, OnSwndWvZ)
      ON_COMMAND(ID_SWND_WV_DIS, OnSwndWvDis)
      ON_COMMAND(ID_SWND_WV, OnSwndWv)
      ON_COMMAND(ID_DYN_STOP, OnDynStop)
      ON_COMMAND(ID_DYN_RESTART, OnDynRestart)
      ON_COMMAND(ID_DYN_CLEAR, OnDynClear)
      ON_WM_DESTROY()
      ON_COMMAND(ID_DISPLAY_CC_PANEL, OnDisplayCcPanel)
      ON_WM_LBUTTONDBLCLK()
      ON_COMMAND(ID_ANALYSIS_START, OnAnalysisStart)                    // 2
      //}}AFX_MSG_MAP
      // 標準印刷コマンド
      ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
      ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
      ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)

      ON_MESSAGE(WM_THREADFINISHED, OnMessageWind)
      ON_MESSAGE(IDS_MESSAGE_WIND, OnMessageWind_G)
END_MESSAGE_MAP()
        .
        .
```

```
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    動的解析スレッド：
//    ―――――――――――――――――――――――――――――――――――――――――――――
UINT threadprocx(LPVOID pParam)                                                    // 3
{
       ierr_dat=0;
       //    ―――――――――――――――――――――――――――――――――――
       //      ●    動的解析用 FORTRAN サブルーチンを呼ぶ
       //    ―――――――――――――――――――――――――――――――――――
       SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
              &n_step,&ns_step,&d_max_v,&id_max_v,
              &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
              &F_read_spring,F_fay,F_n_spring,F_my_spring,F_mz_spring,F_stat_spring,&n_iterate,
              &nm_iterate,&numb_method);
       F_time_ii = ns_step-1 ;
       F_Time=T_analysis;
       //    ―――――――――――――――――――――――――――――――――――
       //      ●    1 回分の解析終了：終了メッセージを送る
       //    ―――――――――――――――――――――――――――――――――――
    ::PostMessage((HWND) pParam, WM_THREADFINISHED, 0,0);
       return 0;
}

//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    動的解析用スレッド「threadprocx」を発生させる。
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CSf3stView::OnAnalysisGo()
{
       icontrol=1;                        // 1:解析実行中パラメータ
    ierr_dat=0;
       // 動的解析スレッド発生
       CWinThread* pThread =
              AfxBeginThread(threadprocx, GetSafeHwnd(),THREAD_PRIORITY_NORMAL);     // 4
}
void CSf3stView::OnAnalysisStart()
{
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    解析中におけるこのコマンドの実行排除
//    ―――――――――――――――――――――――――――――――――――――――――――――
if(thread_on == 1 ) {
       MessageBox("現在計算中です。このコマンドは無視します。");
       return;
}
if(thread_on == 2 ) {
       MessageBox("現在計算停止中です。計算を中止した後、実行して下さい。");
       return;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    固有値解析開始：マルチスレッド処理開始
//    ―――――――――――――――――――――――――――――――――――――――――――――
if(F_calnum == 6){
       MessageBox("固有値解析開始");
       thread_on = 1;
       CWinThread* pThread =
```

```
        AfxBeginThread(threadprocy, GetSafeHwnd(),THREAD_PRIORITY_NORMAL);
        return;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    時刻歴解析開始
//    ―――――――――――――――――――――――――――――――――――――――――
//    ―――――――――――――――――――――――――――――――――
//      ●   予備計算：並列解析の準備
//    ―――――――――――――――――――――――――――――――――
// プロセスを設定する
CDlgSetProcess      dlg;                                                    // 5
if(dlg.DoModal() == IDCANCEL)    return;
// 並列処理の初期化
if(!cMPI_Init(ID_MASTER, (LPCTSTR)dlg.m_strFile))
{
        MessageBox("初期化に失敗しました¥n");
        return;
}
// グローバル変数のセット
icontrol     = 0;          // 予備計算
thread_on    = 1;          // スレッド : 実行中
T_analysis   = 0;          // 解析時間 : 0
nm_iterate   = 0;          //
int nCurrentStep = 0;      // 現在のステップ(local)
// 予備計算 FORTRAN サブルーチンを呼ぶ
SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
        &nCurrentStep,&ns_step,&d_max_v,&id_max_v,
        &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
        &F_read_spring,F_fay,F_n_spring,F_my_spring,F_mz_spring,F_stat_spring,&n_iterate,
        &nm_iterate,&numb_method);
F_time_ii =0;
m_nstep = nCurrentStep;
F_Time=0;
if(ierr_dat != 0)
{
//    ―――――――――――――――――――――――――――――――――
//      ●   予備計算でエラーが発生した
//    ―――――――――――――――――――――――――――――――――
int nx_step;
icontrol = 99;
//    ―――――――――――――――――――――――――――――――――
//      ●   後処理：動的領域の解放を行なう
//    ―――――――――――――――――――――――――――――――――
SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
        &nx_step,&ns_step,&d_max_v,&id_max_v,
        &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
        &F_read_spring,F_fay,F_n_spring,F_my_spring,F_mz_spring,F_stat_spring,&n_iterate,
        &nm_iterate,&numb_method);
//    ―――――――――――――――――――――――――――――――――
//      ●   エラーの原因を表示
//    ―――――――――――――――――――――――――――――――――
err_out(ierr_dat);
if(ierr_dat == 299 )
{
```

```
MessageBox("モデルは解析制限を越えていました。処理を中止します。");
}
else
{
MessageBox("エラーがありました。処理を中止します。エラーの詳細は、表示：動的解析の途中結果の表
示を見てください。");
}
thread_on = 0;
return;
}
//    ―――――――――――――――――――――――――――――――
//      ●　予備計算正常終了
//    ―――――――――――――――――――――――――――――――
//    ―――――――――――――――――――――――――――――――
//      ●　図形処理
//    ―――――――――――――――――――――――――――――――
     Ontime();
     MessageBox("予備計算終了:計算開始");
//    ―――――――――――――――――――――――――――――――
//      ●　解析開始：マルチスレッド処理開始
//    ―――――――――――――――――――――――――――――――
    OnAnalysisGo() ;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●　解析終了処理：解析終了メッセージによって起動される
//    ―――――――――――――――――――――――――――――――――――――――――――――――
LONG CSf3stView::OnMessageWind(UINT wParam, LONG lParam)                    // 6
{
       //    ―――――――――――――――――――――――――――――――
       //      ●　固有値解析の後処理
       //    ―――――――――――――――――――――――――――――――
       if(icontrol == 2){
              //    ―――――――――――――――――――――――――――――――
              //      ●　固有値解析エラーあり
              //    ―――――――――――――――――――――――――――――――
              if(ierr_dat != 0) {
                     if(ierr_dat == 299){
                            MessageBox("モデルは解析制限を越えていました。処理を中止します。");
                     }else{
                            MessageBox("エラーがありました。処理を中止します。");
                     }
                     thread_on = 0;
                     return 1;
              }
              //    ―――――――――――――――――――――――――――――――
              //      ●　固有値解析正常終了
              //    ―――――――――――――――――――――――――――――――
              MessageBox("固有値解析は正常終了しました。");
              thread_on = 0;
       }
       else
       {
              //    ―――――――――――――――――――――――――――――――
              //      ●　時刻歴解析エラーあり
```

```
            //    ―――――――――――――――――――――――――――――――――
            if(ierr_dat != 0) {
                  MessageBox("エラーがありました。処理を中止します。");
                  thread_on = 0;
                  return 1;
            }
            //    ―――――――――――――――――――――――――――――――――
            //      ●  時刻歴解析の後処理
            //    ―――――――――――――――――――――――――――――――――
            if(iend_code == 0){
                  //    ―――――――――――――――――――――――――――――――――
                  //      ●  時刻歴解析一時停止処理
                  //    ―――――――――――――――――――――――――――――――――
                  if(F_holt== 1){
                        Ontime() ;           //  図形処理
                        MessageBox("処理を中断します。");
                  }
                  else
                  {
                        //    ―――――――――――――――――――――――――――――――――
                        //      ●  次ステップの時刻歴解析を実行
                        //    ―――――――――――――――――――――――――――――――――
                        OnAnalysisGo() ;      //  解析実行                    7
                        Ontime() ;            //  図形処理
                  }
            }else{
                  //    ―――――――――――――――――――――――――――――――――
                  //      ●  時刻歴解析正常終了
                  //    ―――――――――――――――――――――――――――――――――
                  Ontime() ;            //  図形処理
                  MessageBox("動的解析は正常終了しました。");
                  thread_on = 0;
                  // 時間計測結果をファイルに書き出す
                  time_log("timelog.txt");
            }
      }
      return 1;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●  解析中止処理
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CSf3stView::OnDynClear()                                          // 8
{
      char buffer[100];
      if(thread_on == 0) {
            MessageBox("初期設定してください。このコマンドは無視します。");
            return;
      }
      if(thread_on == 1) {
            MessageBox("計算実行中です。計算を停止した後実行して下さい。");
            return;
      }
      if(thread_on == 2) {
            CDig_yesno yesno;
```

```
            sprintf(buffer," 計算を中止し、後処理を実行しますか？");
            yesno.m_static_yesno = buffer;
            UpdateData(FALSE);
            if(yesno.DoModal() == IDOK){
                  F_holt= 0;
                  thread_on = 0;
                  int nx_step;
                  icontrol = 98;
            SUBMAIN_DYNAMIC_A(&F_calnum,&iend_code,&icontrol,&ierr_dat,&T_analysis,&dt_analysis,
                        &nx_step,&ns_step,&d_max_v,&id_max_v,
                        &F_read_disp,F_disp,&F_read_ndbalanceF,F_ndbalanceF,
                        &F_read_spring,F_fay,F_n_spring,F_my_spring,F_mz_spring,
                        F_stat_spring,&n_iterate,&nm_iterate,&numb_method);
                  MPI_Finalize();
                  MessageBox("計算を中止し、後処理を実行しました。");
            }
      }
}
```

解析の順序にしたがって、上記のコードを説明しよう。なお、説明文の中の数字は、プログラムコードの右に付したコメント番号である。

処理の多くは、メッセージを受けることによって始まる。このクラスのメッセージは、1)の BEGIN_MESSAGE_MAP() と END_MESSAGE_MAP()で挟まれた関数で受け取ることになる。解析開始のメッセージは、2)の

```
ON_COMMAND(ID_ANALYSIS_START, OnAnalysisStart)
```

で受け取ると、関数 OnAnalysisStart()に制御を移す。この場合のメッセージは、ID_ANALYSIS_START である。

関数 OnAnalysisStart()に制御が移ると、初期設定で thread_on 変数はゼロとしているため、最初の方のコードがスキップされ、制御は、6)の予備計算・並列解析の準備へと移っていく。最初は、PC クラスターを構築するためのファイル設定を行う。先に示したように、まず、クラス CdlgSetProcess のオブジェクトとして、dlg を発生させ、モーダルダイアログを表示させる。ダイアログの中でデータを設定した後、並列構築用のプロセス設定ファイルを作成する。次に、MPI_Init()の拡張版である初期化関数 cMPI_Init()をコールし、この関数の中で分散並列システム（PC クラスター）を構築する。この処理が成功すると次のステップに進むが、失敗するとエラーメッセージを表示した後、この関数から抜けることになる。

並列システム構築に成功すると、解析を制御するグローバル変数に値をセットした後、関数 SUBMAIN_DYNAMIC_A()がコールされる。この関数

thread_on 変数は、マルチスレッドの状態を表す
0：シングルスレッド状態
1：マルチスレッド状態

ここで PC クラスターを構築するためのファイル設定が行われ、cMPI_Init()で、具体的に PC クラスターが構築される。これらの処理は後節で解説する。

ここで、スレーブ側の解析システムが起動される。スレーブ側の解析システムは、初期設定した後、マスター側から解析パラメータや構造データなどを受け取る準備をする。

は、Fortran で書かれたサブルーチンであり、ここでは解析に必要となるデータを読み込み、部材の長さや線形剛性を求めるなどの予備計算を行う。この予備計算処理中に、解析用コントロールデータ、構造データなどの全ての情報をスレーブ側に送信する。各スレーブはこの情報を受信した後、予備計算を始めることになる。

マスター側からスレーブ側に各種の解析に必要なデータが転送される。この処理は、双方共に Fortran 言語で記述されている

入力データなどでエラーがあると、この関数を抜けてきたとき、変数 ierr_dat にエラーコードが設定されており、そのコードに従って処理が選択される。そこでは、まず解析コード icontrol を 99 として、再度 Fortran の関数（実際はサブルーチン）SUBMAIN_DYNAMIC_A()をコールし、この中で動的に確保した領域を解放する。次に、関数 err_out()でエラーコード ierr_dat にしたがってエラー情報をファイルに出力する。同じく、画面にエラー表示を行い、その後、この関数を抜ける。

入力データにエラーがない場合は、解析が進むことになる。まず、図形表示を行うために、関数 Ontime()をコールする。ここでは、子ウインドウで構造図などが表示されている場合は、図形が更新されることになる。その後、予備計算がエラーなく終了したことを画面に表示し、続いて、OnAnalysisGo()関数を用いて、マルチスレッド処理で解析を進めることになる。

この段階は、マスター側とスレーブ側、共に予備計算が終了し、動的解析を実行する手前である

関数 OnAnalysisGo()では、解析実行パラメータ icontrol を 1 にセットした後、4)に示す新たなスレッドを関数 AfxBeginThread()で発生させる。この新しいスレッドは 3)に示す関数 threadprocx()を実行する。ここでは、一回分の解析を実行するために、関数 SUBMAIN_DYNAMIC_A()がコールされる。ここで云う一回の解析とは、動的解析の中の増分解析一回に対応するのではなく、ユーザーが設定する画面描画出力間隔に対応する。例えば、この値が 10 ステップとすると、増分解析が 10 回行われることになる。1 回分の解析が終了して、この関数から制御が戻ってくると、終了メッセージ WM_THREADFINISHED を送出し、この関数を抜けた時点で、発生させたスレッドが終了する。

図形描画を行い、しかも、ユーザーからのメッセージを受信可能にする仕組みが、マスター側のマルチスレッド処理である

この終了メッセージ WM_THREADFINISHED は、メッセージマップ内の関数

```
ON_MESSAGE(WM_THREADFINISHED, OnMessageWind)
```

によって受け取られ、6)に示す関数 OnMessageWind()が実行されることになる。この関数は一回の解析が終了した後、後処理を実行する。解析種別を表す icontrol 変数が 2 の場合は固有値解析であり、2 以外は通常の動的解析処理である。各種の解析制御変数によって処理が異なるが、

動的解析の続きは、7)の関数 OnAnalysisGo()が再度実行され、ここで新たなスレッドを発生させる。処理が戻ってくると関数 Ontime()で、図形の再描画が行われた後、この関数を抜ける。これら一連の処理で動的解析は実行されるわけである。

最後の関数である OnDynClear()は、解析途中で処理を中止した後、動的領域を解放するなどの後処理を行う。この関数は、メッセージマップの中で、

```
ON_COMMAND(ID_DYN_CLEAR, OnDynClear)
```

によってメッセージを受け取り、実行される。ここでは、どのような状態でこの関数がコールされたかをチェックし、その後、関数 SUBMAIN_DYNAMIC_A()をコールして、後処理を行うことになる。

解析終了、もしくは途中で解析中止をした場合、ここで後処理を実行する

3.3 並列システムの構築

本節では、分散並列型システムを構築する過程を詳細に検討しよう。前節で示したように、分散並列型システムの構築は、次に示すように関数 cMPI_Init()によって行われる。この関数は、クラス CMPI_C_Core の初期化関数 Init()をコールするのみである。

```
// 並列処理の初期化
if(!cMPI_Init(ID_MASTER, (LPCTSTR)dlg.m_strFile))
{
	MessageBox("初期化に失敗しました¥n");
	return;
}
```

クラス CMPI_C_Core 全体については、後章で詳細に説明するが、ここでは、必要な部分について簡単に説明する。ファイル mpi_func.cpp の中で、最初に以下のように２つのグロバールオブジェクトが作られている。その後には、各種の関数の定義が続くが、その中で、ここで使用している初期化関数 cMPI_Init()が見られる。

```
// システムの実体
CMPI_C_Core core;
// 一時受信ステータスの実体
MPI_Status system_start;
bool cMPI_Init(int nID, const char* szParam)
{
	return core.Init(nID, szParam);
}
```

ここで、２つのグローバルオブジェクトが作られる

上記の初期化関数で使用している CMPI_C_Core クラスの初期化関数について説明しよう。まず、クラス CMPI_C_Core の初期化関数とヘッダーファイルを以下に示す。分散並列型システムの構築は、ソケットを利用してこの関数が行うことになる。ただし、スレーブとなる各 PC は、デーモンが既に起動され、データの受信待ち受け状態である必要がある。そのため、分散並列システムを構築する場合は、各スレーブ側 PC でデーモンを起動しておかなければならない。

```
// MPI_C_Core.h: CMPI_C_Core クラスのインターフェイス
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MPI_C_CORE_H__1C6F0D42_BEFB_4CC1_B2FF_AAF2169ED5F4__INCLUDED_)
#define AFX_MPI_C_CORE_H__1C6F0D42_BEFB_4CC1_B2FF_AAF2169ED5F4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <winsock2.h>
#include "sock_func.h"
#include "mpi_c.h"

class CMPI_C_Core
{
public:
  CMPI_C_Core();
  virtual ~CMPI_C_Core();
  // 初期化・終了
  bool Init(int nID, const char* szParam);                // mpi_c の初期化処理
  void ShutDownAll();                                     // 全ての接続を切断する
  // 送受信
  inline int Send(int nID, const void* pBuf, int nSize);  // データの送信を行う
  inline int Recv(int nID, void* pBuf, int nSize);        // データの受信を行う
  int BCast(int nRoot, void* pBuf, int nSize);            // データの一斉送信を行う
  // ファイル読み込み
  bool ReadProcess(const char* szPass);                   // プロセス設定ファイルの読み込みを行う
  // 状態取得
  int  GetProcessCount() const;                           // 起動しているプロセスの数を取得
  int GetMyID() const;                                    // 自分のプロセス番号を取得
  int GetRange_S(int nID) const { return m_pInfo[nID].m_nRange[0]; }   // 指定 ID の担当範囲(開始)を取得
  int GetRange_E(int nID) const { return m_pInfo[nID].m_nRange[1]; }   // 指定 ID の担当範囲(終了)を取得
  PROCESS_INFO GetInfo(int i) { return m_pInfo[i]; }      // 指定 ID のプロセス情報を取得
  // 内部初期化処理
private:
      bool SendBoot(SOCKET hSocket, int nID);      // デーモンに起動要求を送信する
      bool SendProcessInfo(SOCKET hSocket);        // プロセス情報を送信
      bool RecvProcessInfo(SOCKET hSocket);        // プロセス情報を受信
      // オブジェクト
public:
      SOCKET  m_pSocket[MAX_PROCESS];              // ソケットハンドル[]
```

```
private:
	PROCESS_INFO m_pInfo[MAX_PROCESS];		// プロセス情報[]
	Int m_nProcess;					// プロセス数
	Int m_nMyID;					// 自分の ID
	char   m_szSlave[PARAM_SIZE];			// スレーブの実行ファイル名(拡張子も含む)
	// 計測用
public:
	int Time_Start(int i = 0);			// 時間の計測を開始
	double Time_Stop(int i = 0);			// 計測時間を取得
	bool TimeLog(const char *szLogFile);		// 計測ログを記録する
private:
	LONGLONG m_llFreq;				// カウンタ周波数
	LONGLONG m_llCnt[MAX_TIMER][2];			// 時間
	LONGLONG m_llAllCnt[MAX_TIMER];			// 合計時間
};

int CMPI_C_Core::Send(int nID, const void* pBuf, int nSize)
{
	return sock_send(m_pSocket[nID], pBuf, nSize);
}

int CMPI_C_Core::Recv(int nID, void* pBuf, int nSize)
{
	return sock_recv(m_pSocket[nID], pBuf, nSize);
}

#endif // !defined(AFX_MPI_C_CORE_H__1C6F0D42_BEFB_4CC1_B2FF_AAF2169ED5F4__INCLUDED_)
// ======================================================================
// CMPI_C_Core::Init
// 概要　: mpi_c の初期化を行う。プロセス情報を元に全てのプロセスを接続を行う。
//		初期化の基本的な流れは以下の通りである。
//		①プロセス情報の取得
//			master: 設定ファイルを読み込む
//			slave : 一つ前のプロセスと接続し、プロセス情報を受け取る。
//		②接続アルゴリズム
//			自分より小さい ID		: 接続する
//			自分より一つ大きい ID		: 起動させ、接続を待ち受ける
//			自分より大きい ID		: 相手からの接続を待ち受ける
//	※　起動させるプロセスでは mpi_c デーモンが起動している必要がある。
// 引数　:　nID		= 自分の ID
//		szParam		= パラメータ引数
//				master: プロセス設定ファイルのフルパス,
//				slave: 一つ前の ID のコンピュータ名(接続用に用いる)
// 戻り値: bool	: 他のプロセスとの接続に失敗した場合は false が返る
// ======================================================================
bool CMPI_C_Core::Init(int nID, const char* szParam)
{
	int i;			// カウンタ
	SOCKET hDeamon;		// デーモン接続用ソケット
	SOCKET hSocket;		// ソケットの一時保存用
	// Winsock の初期化
	if(sock_init() == false)					// 1
	{
		return false;
```

```
}
// マスター(nID = 0)なら、プロセス設定ファイルを読み込む
if(nID == ID_MASTER)
{
	ReadProcess(szParam);                                          // 2

}
// それ以外なら、自分の前のプロセスにアクセスして、プロセス情報を受け取る
else
{
	hSocket = sock_connect(szParam, PORT_PARALLEL + (nID -1));     // 3
	if(hSocket == INVALID_SOCKET)   return false;
	RecvProcessInfo(hSocket);
}
// 自分の ID を設定
m_nMyID = nID;                                                     // 4

// 接続開始
for(i=0; i < m_nProcess; i++)                                      // 5
{
	// 自分のひとつ前のプロセスは無視(既に接続しているので)
	if(i == m_nMyID -1)                                            // 6
	{
		m_pSocket[i] = hSocket;
	}
	// 自分より小さいプロセスには接続
	else if(i < m_nMyID)                                           // 7
	{
		m_pSocket[i] = sock_connect(m_pInfo[i].m_szHost, PORT_PARALLEL + i);
		if(m_pSocket[i] == INVALID_SOCKET)
		{
			return false;
		}
	}
	// 自分の場合は待ち受け開始
	else if(i == m_nMyID)                                          // 8
	{
		// クライアントからの待ち受け用ソケットを作成
		m_pSocket[i] = sock_listen(PORT_PARALLEL + m_nMyID);
		if(m_pSocket[i] == INVALID_SOCKET)
		{
			return false;
		}
	}
	// 自分より１つ大きいプロセスは起動させて、接続を待つ
	else if(i == m_nMyID + 1)                                      // 9
	{
		// デーモンと接続し、起動要求を出す
		hDeamon = sock_connect(m_pInfo[i].m_szHost, PORT_DEAMON);
		SendBoot(hDeamon, i);
		sock_close(hDeamon);
		// 自分より１つ大きいプロセスからの接続を待つ
		if(sock_accept(m_pSocket[m_nMyID], m_pSocket[i]) == false)
		{
```

```
                        return false;
                    }
                    // プロセス情報を送信
                    SendProcessInfo(m_pSocket[i]);
                }
                // 自分より大きいプロセスは接続を待つ
                else                                                    // 10
                {
                    if(sock_accept(m_pSocket[m_nMyID], m_pSocket[i]) == false)
                    {
                        return false;
                    }
                }
            }
            return true;
}
```

この関数は、マスターとスレーブの両者で使用する。そこで、両者別々にその動作を検証しよう。まず、マスター側の動きを観察する。

1. 関数 sock_init()により、winsock の初期化を行う。ソケット関数を使用する場合は、必ずこの関数を実行しなければならない。ここで初期化に失敗すると false を関数値として、この関数から戻ることになる。
2. プロセス ID（このマニュアルではランクと同一の意味で用いている）が ID_MASTER に等しいとき、関数 ReadProcess()によりプロセス設定ファイルを読み込む。
3. ここは、スレーブ側の処理となる。
4. メンバー変数 m_nMyID に自分のプロセス ID をセットする。
5. ここからは、全プロセスについて接続を開始する。その処理内容は、自分のプロセス ID と他との関係によって処理が異なる。ここで用いているプロセス総数 m_nProcess は、関数 ReadProcess()でプロセス設定ファイルが既に読まれており、セットされている。
6. ここは、マスター側では起こり得ない。
7. ここは、マスター側では起こり得ない。
8. プロセス ID が 0 の場合、つまりマスターにおける処理はスレーブからの待ち受け用のソケットを作成する。関数 sock_listen()により、他の PC からの接続を待ち受ける。この関数からは、ソケットのハンドル値が戻される。これを配列 m_pSocket[i]にセットする。接続に失敗すると false を関数値として、この関数から戻ることになる。
9. プロセス ID が 1 のスレーブに対しては、つまり、自分の ID よりひとつ数字の大きいスレーブが動作する PC に対し、次の処理を行う。

マスター:プロセス ID:0
スレーブ:プロセス ID: １以上の値

まず、関数 sock_connect()で該当の PC と接続し、関数戻り値として接続ハンドルを hDeamon にセットする。接続用ポートとして PORT_DEAMON を用いる。次に、関数 SendBoot()で、デーモンにスレーブ用の動的解析システムを起動させ、その後、sock_close()を用いて接続用ソケットを閉鎖する。次に、起動させたスレーブが関数 sock_accept()を用いて、マスターに接続してくるのを待つ。接続があるまで処理はブロッキングされる。接続があると、関数 SendProcessInfo()を用いて、プロセス設定情報を送信する。

10. この条件はプロセス ID が 1 以外の全スレーブに対し成立し、以下の処理が行われる。関数 sock_connect()を用いてスレーブとの接続が行われる。この関数の引数は、スレーブ側の PC 名が IP アドレス、接続したポート番号である。接続に成功するとソケットのハンドル値が戻され、これを配列 m_pSocket[i]にセットする。接続に失敗すると false を関数値として、この関数から戻ることになる。

以上でマスターにおける初期化処理が終了する。ここでは全スレーブとの接続を行い、プロセス ID が 1 に対しては、デーモンにスレーブ用動的解析システムを起動させ、そのスレーブにプロセス情報を送信する。

次に、スレーブの初期化処理について説明しよう。スレーブは、当該 PC 上のデーモンによって起動させられ、処理が始まることになる。この処理の初めで、初期化関数が実行されることになる。

1. 関数 sock_init()により、winsock の初期化を行う。ソケット関数を使用する場合は、必ずこの関数を実行しなければならない。ここで初期化に失敗すると false を関数値として、この関数から戻ることになる。
2. ここは、マスター側の処理となる。
3. 自分のプロセス ID より、ひとつ前の ID を有するプロセスに、関数 sock_connect()を用いて接続する。このひとつ前のプロセスは、マスター、スレーブを問わない。この後、このプロセスからプロセス設定情報を関数 RecvProcessInfo()を用いて受け取ることになる。
4. メンバー変数 m_nMyID に自分のプロセス ID をセットする。
5. ここからは、全プロセスについて接続を開始する。その処理内容は、自分のプロセス ID と他との関係によって処理が異なる。ここで用いているプロセス総数 m_nProcess は、関数 ReadProcess()で、プロセス設定ファイルを読むことで既にセットされている。

6. 一つ前のプロセス ID に対しては、既に 3. で接続されているので、ここでは、そこで得た接続ハンドルを配列 m_pSocket[i]にセットする。
7. 自分より小さいプロセス ID に対して、関数 sock_connect()で該当の PC と接続し、関数戻り値として接続ハンドルを m_pSocket[i]にセットする。
8. プロセス ID が i の場合、つまり自分自身における処理はスレーブからの待ち受け用のソケットを作成する。関数 sock_listen()により、他の PC からの接続を待ち受ける。この関数からは、ソケットのハンドル値が戻される。これを配列 m_pSocket[i]にセットする。接続に失敗すると false を関数値として、この関数から戻ることになる。
9. プロセス ID が i+1 のスレーブに対しては、つまり、自分の ID よりひとつ数字の大きいスレーブが動作する PC に対し、次の処理を行う。まず、関数 sock_connect()で該当の PC と接続し、関数戻り値として接続ハンドルを hDeamon にセットする。接続用ポートとして PORT_DEAMON を用いる。次に、関数 SendBoot()で、デーモンにスレーブ用の動的解析システムを起動させ、その後、sock_close()を用いて接続用ソケットを閉鎖する。次に、起動させたスレーブが関数 sock_accept()を用いて、マスターに接続してくるのを待つ。接続があるまで、処理はブロッキングされる。接続があると、関数 SendProcessInfo()を用いて、プロセス設定情報を送信する。
10. この条件はプロセス ID が i+1 以上の全スレーブに対し成立し、以下の処理が行われる。関数 sock_connect()を用いて、スレーブとの接続が行われる。この関数の引数は、スレーブ側の PC 名が IP アドレス、接続したポート番号である。接続に成功するとソケットのハンドル値が戻され、これを配列 m_pSocket[i]にセットする。接続に失敗すると false を関数値として、この関数から戻ることになる。

以上で、スレーブにおける初期化処理が終了する。ここでは、マスター及び他のスレーブとの接続を行い、さらに、ひとつ小さいプロセス ID を有するプロセスからプロセス情報を受信する。また、プロセス ID が自分のプロセス ID よりひとつ大きいプロセスに対して、デーモンにスレーブ用動的解析システムを起動させ、そのスレーブにプロセス情報を送信する。これで、マスターを含めた全プロセスが、他のプロセスと接続を行い、そのソケットハンドルとプロセス設定情報を取得したことになる。

3.4 並列システム設定ダイアログ

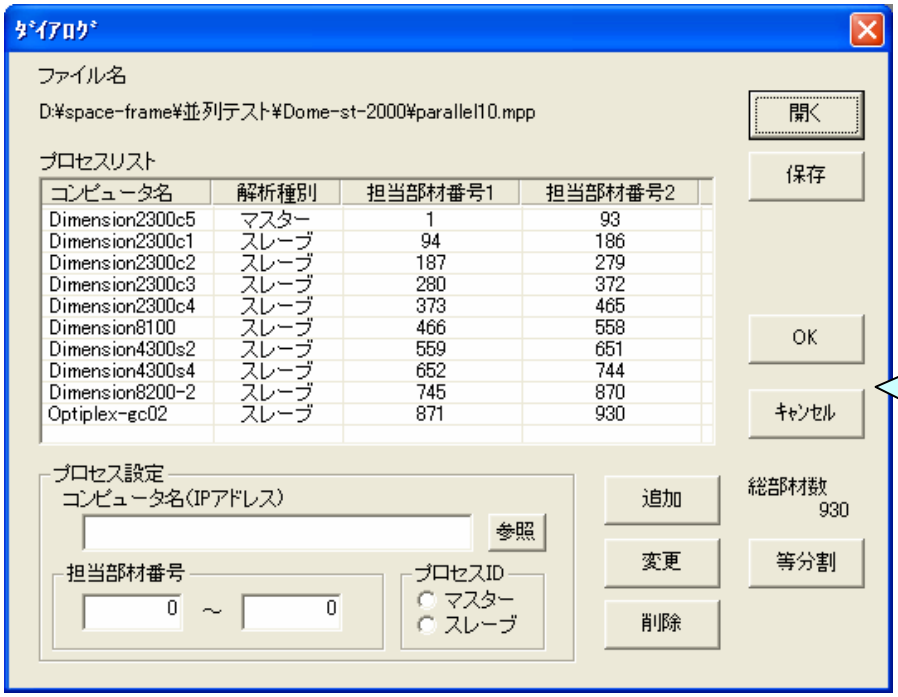
本節では、プロセス設定ファイルを作成する処理について、詳細に解説する。分散並列型動的解析を行うために、まず、システムは PC クラスターを構築する。その際、以下に示すようなプロセス設定ファイルが必要となる。

```
sf3pa_slave.exe
Dimension2300c5     1     93
Dimension2300c1     94    186
Dimension2300c2     187   279
Dimension2300c3     280   372
 .
```

プロセス設定用ファイルは、当該のコントロールファイルと同じフォルダー内に存在しなければならない

このファイルの仕様は、第１行目がスレーブ側で実行するプログラム名、第２行目以降が並列システムを構築するためのパラメータである。各行で、第一項が PC 名もしくは IP アドレス、第二項がその PC が分担する部材の最初の部材番号、第三項が最終部材番号を示す。また、PC 用のパラメータで、第１行の PC は、マスターで、それ以降はスレーブとなるように設定されている。

このファイルを用いて、初期化関数 cMPI_Init()によって、スレーブとなる各 PC に起動が掛けられ、上記のファイルに書き込まれたスレーブ用のプログラムが実行される。



OK ボタンは、実行開始となり、キャンセルボタンは、解析の実行も中止となる。

図 3-2 並列システム構築用ダイアログ

次に、このファイルをどのように設定するかについて説明しよう。まず、並列システムを構築するためのダイアログが図 3-2 に示される。こ

のダイアログの中で、データを設定するための機能について説明する。ダイアログ右側の「OK」ボタンを押すと、ダイアログ設定から抜け、初期化関数 cMPI_Init()を実行する。同様に「キャンセル」ボタンでは、ダイアログ設定を終了するが、そのまま何もせずにこの関数を抜けることになる。その際、動的解析開始もキャンセルされることになる。

既に、設定ファイルが存在する場合は、「開く」ボタンを押し、ファイルからデータを入力する。また、設定したデータをファイルに保存したい場合は、「保存」ボタンを押下する。

次に、ダイアログを用いてデータを設定する方法について説明する。ダイアログ下側の右のボタンは、「追加」、「変更」、「削除」であり、これらのボタンの左側のデータ入力領域と上のプロセス表示領域に対して、上記機能が実行される。最初の「追加」ボタンは、左側のデータ入力領域で設定したデータをプロセス表示領域の最後に追加する。先にプロセス表示領域で PC を指定すると、データ入力領域にそのデータがコピーされてセットされる。そこで、データを変更し、「変更」ボタンを押すと、データが変更されることになる。プロセス表示領域で PC を指定した後、「削除」ボタンを押すと、その指定した PC データが削除され、プロセス表示領域はその部分を取り除いて表示される。

データ入力領域では、コンピュータ名あるいは IP アドレス、担当部材の最初と最後の部材番号、マスターかスレーブの設定を行う。コンピュータ名の設定は、「参照」ボタンを押すことでネットワーク内を探索することができ、該当コンピュータを指定することができる。解析モデルの総部材数が表示されているので、この範囲内で各 PC の担当部材を設定することになる。ここで、「等分割」ボタンを押すと、それまでに設定したマスターと各スレーブに対し、担当部材数を等しく分割し、その値をプロセス表示領域に表示する。

ここで設定したデータの矛盾や間違いは、重大なエラーを引き起こし、ほとんど場合、システムはハングする。SPACE システム内でもデータチェックを行っているが、ユーザーもデータを十分に検証されたい。

このダイアログクラスは、DlgSetProcess で実行されており、以下にその内容を示す。難しいロジックを含まないので、コメントなどを参照することで理解できると思う。

```
//    ――――――――――――――――――――――――――――――――――――――――
// DlgSetProcess.cpp : インプリメンテーション ファイル
//    ――――――――――――――――――――――――――――――――――――――――
//      ●    DlgSetProcess.cpp : インプリメンテーション ファイル
//    ――――――――――――――――――――――――――――――――――――――――
```

```
#include "stdafx.h"
#include "sf3stdatex.h"
#include "sf3st.h"
#include "DlgSetProcess.h"
#include <fstream.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　項目名の設定
//    ―――――――――――――――――――――――――――――――――――――――――
LPTSTR szHead[] =
{
      "コンピュータ名", "解析種別","担当部材番号 1", "担当部材番号 2", NULL
};
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　CDlgSetProcess ダイアログの構築
//    ―――――――――――――――――――――――――――――――――――――――――
CDlgSetProcess::CDlgSetProcess(CWnd* pParent /*=NULL*/)
      : CDialog(CDlgSetProcess::IDD, pParent)
{
      //{{AFX_DATA_INIT(CDlgSetProcess)
      m_nMem = 0;
      m_nChargeMem_e = 0;
      m_nChargeMem_s = 0;
      m_strHost = _T("");
      m_strFile = _T("");
      m_radID = -1;
      //}}AFX_DATA_INIT
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　データのチェンジ
//    ―――――――――――――――――――――――――――――――――――――――――
void CDlgSetProcess::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CDlgSetProcess)
      DDX_Control(pDX, IDC_LIST, m_ListCtrl);
      DDX_Text(pDX, IDC_EDIT_MEM, m_nMem);
      DDX_Text(pDX, IDC_EDIT_MEM2, m_nChargeMem_e);
      DDX_Text(pDX, IDC_EDIT_MEM1, m_nChargeMem_s);
      DDX_Text(pDX, IDC_EDIT_HOST, m_strHost);
      DDX_Text(pDX, IDC_EDIT_FILE, m_strFile);
      DDX_Radio(pDX, IDC_RAD_ID, m_radID);
      //}}AFX_DATA_MAP
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　メッセージマップ
//    ―――――――――――――――――――――――――――――――――――――――――
BEGIN_MESSAGE_MAP(CDlgSetProcess, CDialog)
      //{{AFX_MSG_MAP(CDlgSetProcess)
      ON_NOTIFY(NM_CLICK, IDC_LIST, OnClickList)
```

```
        ON_BN_CLICKED(IDC_ADD_LIST, OnAddList)
        ON_BN_CLICKED(IDC_DELETE_LIST, OnDeleteList)
        ON_BN_CLICKED(IDC_EDIT_PROCESS, OnEditProcess)
        ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
        ON_BN_CLICKED(IDC_OPEN, OnOpen)
        ON_BN_CLICKED(IDC_SAVE, OnSave)
        ON_BN_CLICKED(IDC_EQUQL_PART_MEMBER, OnEquqlPartMember)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ダイアログの初期化
//    ―――――――――――――――――――――――――――――――――――――――
BOOL CDlgSetProcess::OnInitDialog()
{
        CDialog::OnInitDialog();
        m_nMem= mnbsb;                          // 総部材数を表示
        m_bDirty = FALSE;                       // 更新フラグ
        // 項目名を設定
        SetHeader(szHead);
        // リストアイテムの１行全選択, 罫線表示モード
        m_ListCtrl.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES );
        // リストビューを更新
        UpdateList();
        return TRUE;  // コントロールにフォーカスを設定しないとき、戻り値は TRUE となります
                      // 例外: OCX プロパティ ページの戻り値は FALSE となります
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　リストの表示
//    ―――――――――――――――――――――――――――――――――――――――
void CDlgSetProcess::OnClickList(NMHDR* pNMHDR, LRESULT* pResult)
{
        CString strID;      // ID の入った文字列
        int nIndex;
        POSITION pos;
        PROCESS_INFO* pInfo;
        // 現在選択されているリストのインデックスを取得
        nIndex = m_ListCtrl.GetSelectionMark();
        // 何も選択されていない場合は無視
        if(nIndex == -1)    return;
        // 選択されたプロセスをエディットボックスに表示
        pos = m_listInfo.FindIndex(nIndex);
        if(pos == NULL)     return;
        pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
        m_strHost = pInfo->m_szHost;
        if(pInfo->m_nID == ID_MASTER)
                m_radID = 0;
        else
                m_radID = 1;
        m_nChargeMem_s = pInfo->m_nRange[0];
        m_nChargeMem_e = pInfo->m_nRange[1];
        // ダイアログを更新
        UpdateData(FALSE);
        *pResult = 0;
}
```

```
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　リストの更新
//    ―――――――――――――――――――――――――――――――――――――――――
// ==========================================
// CDlgSetProcess::UpdateList
// 概要　: リストコントロールの更新
// 引数　: none
// 戻り値: none
// ==========================================
void CDlgSetProcess::UpdateList()
{
	POSITION pos;
	PROCESS_INFO* pTemp;
	int i = 0;
	// リストの全アイテムを削除
	m_ListCtrl.DeleteAllItems();
	// プロセス情報をリストにセット
	for(pos = m_listInfo.GetHeadPosition(); pos != NULL; m_listInfo.GetNext(pos))
	{
		pTemp = (PROCESS_INFO*)m_listInfo.GetAt(pos);
		SetList(pTemp, i++);
	}
	// ダイアログを更新
	UpdateData(FALSE);
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　リストビューにデータをセット
//    ―――――――――――――――――――――――――――――――――――――――――
// ==========================================
// CDlgSetProcess::SetList
// 概要　: リストビューにデータをセット
// 引数　: pElem　= セットする梁要素オブジェクトのポインタ
//           : num　　= アイテムのインデックス
// 戻り値: none
// ==========================================
void CDlgSetProcess::SetList(PROCESS_INFO* pInfo, int num)
{
	LVITEM item;
	int nIndex = 0;
	// メインアイテムにデータをセット
	item.mask		= LVIF_TEXT; // pszText(項目名文字列)
	item.iItem		= num;		// 行番号
	item.iSubItem	= 0;		// 列項目
	item.pszText =  pInfo->m_szHost;// マシン名
	nIndex = m_ListCtrl.InsertItem(&item);
	// サブアイテムにデータをセット
	m_ListCtrl.SetItemText(nIndex, 1, GetProcessType(pInfo->m_nID));// プロセスタイプ
	char buf[10];
	_itoa(pInfo->m_nRange[0], buf, 10);
	m_ListCtrl.SetItemText(nIndex, 2, buf);		// 担当部材[開始]
	_itoa(pInfo->m_nRange[1], buf, 10);
	m_ListCtrl.SetItemText(nIndex, 3, buf);		// 担当部材[終了]
	// ダイアログを更新
	UpdateData(FALSE);
```

```
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ロセスタイプの取得
//    ―――――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::GetProcessType
// 概要　: ID からプロセスタイプ(Master,Slave)を取得する
// 引数　: nID      = プロセスの ID
// 戻り値: LPCTSTR : ID に応じてプロセス名が返る
// ========================================
LPCTSTR CDlgSetProcess::GetProcessType(int nID)
{
      CString buf;
      switch(nID)
      {
      case ID_MASTER:
            return "マスター";
      default:
            return "スレーブ";
      }
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ヘッダーをセット
//    ―――――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::SetHeader
// 概要　: ヘッダーをセット
// 引数　: header[]       = ヘッダーの項目名
// 戻り値: none
// ========================================
void CDlgSetProcess::SetHeader(LPSTR header[])
{
      int i;
      // 項目名をセット
      for(i=0; header[i] != NULL; i++)
      {
            SetColum(header[i], i, m_ListCtrl.GetStringWidth(header[i])+ 20);
      }
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●　ヘッダーの各項目名をセット
//    ―――――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::SetColum
// 概要　: ヘッダーの各項目名をセット
// 引数　: name           = 各項目名
//           : count      = 列番号
//           : range      = 各列のデフォルトの幅(pixcel)
// 戻り値: none
// ========================================
void CDlgSetProcess::SetColum(LPTSTR name, int count, int range)
{
      // カラムの設定
      LVCOLUMN     myColumn;
```

```
	myColumn.mask	= LVCF_TEXT|LVCF_WIDTH|LVCF_FMT;  // pszText, cx, format を有効にする
	myColumn.cx		= range + 10;                      // 横幅の range%を表題にする
	myColumn.fmt	= LVCFMT_CENTER;                   // 表題の中央に項目名を配置
	myColumn.pszText	= name;                        // 列タイトル文字列
	m_ListCtrl.InsertColumn(count,&myColumn);          // count 行に項目名をセット
}
//    ―――――――――――――――――――――――――――――――――――――――――
//     ●　リスト上で選択されている項目の削除
//    ―――――――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::OnDeleteList
// 概要　: 「削除」ボタンが押されたとき
//                リスト上で選択されているソケット情報を削除
// 引数　: none
// ========================================
void CDlgSetProcess::OnDeleteList()
{
	CString strID;	// ID の入った文字列
	int nIndex;
	POSITION pos;
	PROCESS_INFO* pInfo;
	// 現在選択されているスレーブを取得
	nIndex = m_ListCtrl.GetSelectionMark();
	// 何も選択されていない場合は無視
	if(nIndex == -1)	return;
	// 選択されたソケット情報を削除
	pos = m_listInfo.FindIndex(nIndex);
	pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
	m_listInfo.RemoveAt(pos);
	delete pInfo;
	// リストを更新
	UpdateList();
}
//    ―――――――――――――――――――――――――――――――――――――――――
//     ●　情報の更新
//    ―――――――――――――――――――――――――――――――――――――――――
// ================================================================
// CDlgSetProcess::OnEditProcess
// 概要　:　「更新」ボタンを押したとき
//                選択されているプロセスの情報を更新
// 引数　:　none
// 戻り値: none
// ================================================================
void CDlgSetProcess::OnEditProcess()
{
	int nIndex;
	POSITION pos;
	PROCESS_INFO* pInfo;
	// ダイアログのデータを取得
	UpdateData(TRUE);
	// 現在選択されているリストのインデックスを取得
	nIndex = m_ListCtrl.GetSelectionMark();
	// 何も選択されていない場合は無視
	if(nIndex == -1)	return;
```

```
	// 確認メッセージを表示
	if(AfxMessageBox("選択されたプロセスの設定を変更します。¥n"
		"よろしいですか?", MB_OKCANCEL) == IDCANCEL)	return;
	// コンピュータ名のチェック
	if(m_strHost.IsEmpty())
	{
		AfxMessageBox("コンピュータ名が空欄です。");
		return;
	}
	// 選択されたプロセス情報を更新
	pos = m_listInfo.FindIndex(nIndex);
	if(pos == NULL)	return;
	pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
	strcpy(pInfo->m_szHost, (LPCTSTR)m_strHost);
	if(m_radID == 0)
		pInfo->m_nID = ID_MASTER;
	else
		pInfo->m_nID = ID_SLAVE;
	pInfo->m_nRange[0] = m_nChargeMem_s;
	pInfo->m_nRange[1] = m_nChargeMem_e;
	// リストを更新
	m_bDirty = TRUE;		// 更新フラグを TRUE
	UpdateList();
}
//	―――――――――――――――――――――――――――――――――――――
//	●　リストへの追加
//	―――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::OnAddList
// 概要　: 「追加」ボタンが押されたとき
// 引数　: none
// ========================================
void CDlgSetProcess::OnAddList()
{
	PROCESS_INFO* pInfo = NULL;
	BOOL bLocal = FALSE;
	DWORD dwRet = MAX_COMPUTERNAME_LENGTH + 1;
	// ダイアログの値を取得
	UpdateData(TRUE);
	// コンピュータ名のチェック
	if(m_strHost.IsEmpty())
	{
		if(AfxMessageBox("コンピュータ名が空欄です。¥n"
			"現在使用しているコンピュータを設定しますか？", MB_OKCANCEL) == IDCANCEL)
			return;
		// コンピュータ名を取得して設定
		bLocal = TRUE;
	}
	// 新規のプロセス情報を生成
	pInfo = new PROCESS_INFO;
	// エディットボックスの値をコピー
	if(bLocal)
	{
		GetComputerName(pInfo->m_szHost, &dwRet);
```

```
        //    GetLocalName(pInfo->m_szHost, strlen(pInfo->m_szHost));
        }
        else
              strcpy(pInfo->m_szHost, (LPCTSTR)m_strHost);
        if(m_radID == 0)
              pInfo->m_nID = ID_MASTER;
        else
              pInfo->m_nID = ID_SLAVE;
        pInfo->m_nRange[0] = m_nChargeMem_s;
        pInfo->m_nRange[1] = m_nChargeMem_e;
        // リストに追加
        m_listInfo.AddTail(pInfo);
        // リストビューを更新
        m_bDirty = TRUE;                  // 更新フラグを TRUE
        UpdateList();
  }
  //    ―――――――――――――――――――――――――――――――――――――――
  //     ●　参照
  //    ―――――――――――――――――――――――――――――――――――――――
  // ================================================================
  // CDlgSetProcess::OnBrowse
  // 概要　:　「参照」ボタンを押したとき
  //                マイネットワークダイアログからコンピュータ名を取得
  // 引数　:　none
  // 戻り値: none
  // ================================================================
  void CDlgSetProcess::OnBrowse()
  {
        char         chPutMachine[MAX_PATH];          // 選択されたマシン名
        LPITEMIDLIST pidlRetFolder;                   // シェルの戻り値
        BROWSEINFO   stBInfo;                         // シェルブラウザ構造体
        LPITEMIDLIST pIDL;                            // ルートフォルダ ID
        // ルートフォルダをマイネットワークに指定
        SHGetSpecialFolderLocation(GetSafeHwnd(), CSIDL_NETWORK, &pIDL);
        //  構造体を初期化
        stBInfo.pidlRoot = pIDL;                              //ルートフォルダ
        stBInfo.hwndOwner = GetSafeHwnd();           //表示するダイアログの親ウィンドウのハンドル
        stBInfo.pszDisplayName = chPutMachine;                //選択されているマシン名
        stBInfo.lpszTitle = "スレーブを起動させるコンピュータを選択して下さい。";  //説明の文字列
        stBInfo.ulFlags = BIF_BROWSEFORCOMPUTER;              //表示フラグ
        stBInfo.lpfn = NULL;      //ダイアログプロシージャへのポインタ
        stBInfo.lParam = 0L;      //プロシージャに送るパラメーター
        //  ダイアログボックスを表示
        pidlRetFolder = ::SHBrowseForFolder( &stBInfo );
        if( pidlRetFolder != NULL )
        {
              // 取得したコンピュータ名をセット
              if(chPutMachine)
                    m_strHost = chPutMachine;
              ::CoTaskMemFree( pidlRetFolder );
        }
        // ダイアログを更新
        UpdateData(FALSE);
  }
```

```
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　データ保存
//    ―――――――――――――――――――――――――――――――――――――――――
// ========================================
// CDlgSetProcess::OnOpen
// 概要　：「開く」ボタンを押したときの処理
//              ファイルを選択し、読み込む。
// 引数　: none
// 戻り値: none
// ========================================
void CDlgSetProcess::OnOpen()
{
      // 現在の設定を保存するか確認
      if(m_bDirty == TRUE)
      {
            if(AfxMessageBox("現在のプロセス情報を保存しますか？",
                  MB_OKCANCEL) == IDOK)
            {
                  OnSave();
                  return;
            }
      }
// ”オープン”モードのファイル選択ダイアログの生成
      CFileDialog dlg( TRUE, "mpp", NULL,
                       OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
                  "プロセスデータ (*.mpp)|*.mpp|すべてのファイル(*.*)|*.*||", NULL);
      if (dlg.DoModal() == IDOK )
      {
            m_strFile = dlg.GetPathName();  // ファイルのフルパス名を取得
            ReadFile((LPCTSTR)m_strFile);
      }
      // リストを更新
      m_bDirty = FALSE;               // 更新フラグをリセット
      UpdateList();
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　データファイルの入力
//    ―――――――――――――――――――――――――――――――――――――――――
// ============================================================
// CDlgSetProcess::ReadFile
// 概要　：　設定ファイルの読み込み
// 引数　：　szFile= ファイルのフルパス名
// 戻り値: BOOL    : 読み込み成功したら TRUE
// ============================================================
BOOL CDlgSetProcess::ReadFile(LPCTSTR szFile)
{
      int   nCount = 0;
      int   nRead;                    // 読み込んだ要素の数
      char  szBuf[256] = "";          // 読み込みバッファ
      POSITION pos;
// 読み込みデータ
      PROCESS_INFO *pInfo;            // 新規作成するソケット情報
      PROCESS_INFO tempInfo;          // 一時保存用
// ファイルオブジェクト
```

```
        ifstream file(szFile, ios::in);
        if(!file)
        {
              AfxMessageBox("ファイルオープン失敗¥n");
              return FALSE;
        }
        // 前回のリストを消去する
        if(!m_listInfo.IsEmpty())
        {
              pos = m_listInfo.GetHeadPosition();
              for(; pos != NULL; pos = m_listInfo.GetHeadPosition())
              {
                    pInfo = (PROCESS_INFO*)m_listInfo.RemoveHead();
                    delete pInfo;
              }
        }
        // スレーブアプリ名を読み込む
        file.getline(szBuf, 256);
        // ファイル内容を一行ずつ EOF 間で読み出す
        while(!file.eof())
        {
              file.getline(szBuf, 256);              // 1 行読み込み
              if(!szBuf)   continue;                 // 読み込み失敗なら次の行を読み込む
              // ソケット情報の読み込み  ex) "Dimension8200-2 2 1 30"
              nRead = sscanf(szBuf, "%s %d %d¥n",
                                  &tempInfo.m_szHost,
                                  &tempInfo.m_nRange[0], &tempInfo.m_nRange[1]);
              if(nRead != 3)     continue;           // 正常に読み込めなかったら、次の行を読み込む
              // プロセスを生成
              pInfo = new PROCESS_INFO;
              strcpy(pInfo->m_szHost, tempInfo.m_szHost);
              pInfo->m_nID = nCount++;
              pInfo->m_nRange[0] = tempInfo.m_nRange[0];
              pInfo->m_nRange[1] = tempInfo.m_nRange[1];
              // ソケット情報をリストに追加
              m_listInfo.AddTail(pInfo);
        }
        return TRUE;
  }
  //    ―――――――――――――――――――――――――――――――――――――――
  //     ●    データの保存
  //    ―――――――――――――――――――――――――――――――――――――――
  // ============================================================
  // CDlgSetProcess::OnSave
  // 概要  :   「保存」ボタンを押したとき
  //                保存ファイル名を選択して保存
  // 引数  :   szFile= ファイルのフルパス名
  // 戻り値: BOOL     : 読み込み成功したら TRUE
  // ============================================================
  void CDlgSetProcess::OnSave()
  {
  // ”保存”モードのファイル選択ダイアログの生成
        CFileDialog dlg( FALSE, "mpp", NULL,
                         OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
```

```
                    "プロセスデータ (*.mpp)|*.mpp|すべてのファイル(*.*)|*.*||", NULL);
        if (dlg.DoModal() == IDOK )
        {
              m_strFile = dlg.GetPathName();  // ファイルのフルパス名を取得
              SaveFile((LPCTSTR)m_strFile);
        }
        m_bDirty = FALSE;                     // 更新フラグをリセット
        UpdateList();
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　設定ファイルの保存
//    ―――――――――――――――――――――――――――――――――――――――――
// ================================================================
// CDlgSetProcess::SaveFile
// 概要　:　設定ファイルの保存
// 引数　:　szFile= ファイルのフルパス名
// 戻り値: BOOL    : 保存成功したら TRUE
// ================================================================
BOOL CDlgSetProcess::SaveFile(LPCTSTR szFile)
{
        POSITION     pos;
        PROCESS_INFO*       pInfo = NULL;
// ファイルオブジェクト
        ofstream file(szFile, ios::out);
        if(!file)
        {
              AfxMessageBox("ファイルオープン失敗¥n");
              return FALSE;
        }
        // スレーブアプリ名を書き込む
        file << "sf3pa_slave.exe" << endl;
        // プロセス情報を書き込む
        for(pos = m_listInfo.GetHeadPosition(); pos != NULL; m_listInfo.GetNext(pos))
        {
              pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
              file << pInfo->m_szHost << "¥t"
                          << pInfo->m_nRange[0] << "¥t" << pInfo->m_nRange[1] << endl;
        }
        return TRUE;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●　担当メンバーのチェック
//    ―――――――――――――――――――――――――――――――――――――――――
// ================================================================
// CDlgSetProcess::CheckMember
// 概要　:　担当メンバーをチェック
//                  他の場所で重複指定されていないかチェック
// 引数　:　nMem = 部材番号
// 戻り値: BOOL    : 重複がなければ TRUE
// ================================================================
BOOL CDlgSetProcess::CheckMember(int nMem)
{
        POSITION pos = m_listInfo.GetHeadPosition();
        PROCESS_INFO*       pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
```

```
        // 担当部材番号が、部材数を超えていないかチェック
        if(nMem <= 0 || nMem < mnbsb)          return FALSE;
        // 他のリストで選択されている範囲をチェック (開始番号 <= nMem <= 終了番号)のとき
        for(; pos != NULL; pInfo = (PROCESS_INFO*)m_listInfo.GetNext(pos))
        {
              if(nMem >= pInfo->m_nRange[0] && nMem <= pInfo->m_nRange[1])
                    return FALSE;
        }
        return TRUE;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　ファイルのチェック
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CDlgSetProcess::OnOK()
{
        // 最終チェックをする
        if(m_bDirty)
        {
              if(AfxMessageBox("現在の設定は保存されていません。¥n 保存しますか？", MB_OKCANCEL)
                    == IDOK)
              {
                    OnSave();
              }
        }
        // ファイル名が無ければエラー
        if(m_strFile.IsEmpty())
        {
              AfxMessageBox("プロセス設定ファイルが指定されていません。");
              return;
        }
        CDialog::OnOK();
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●　担当部材の等分割処理
//    ―――――――――――――――――――――――――――――――――――――――――――――
// ================================================================
// CDlgSetProcess::OnEquqlPartMember
// 概要　:　部材の「等分割」ボタンを押したとき
//                担当部材を等分割する(余りはマスターに)
// 引数　:　none
// 戻り値: none
// ================================================================
void CDlgSetProcess::OnEquqlPartMember()
{
        int nProcess;                    // プロセス数
        int   nPart;                     // 担当部材数
        int nRemainder;                  // 等分した余り
        int nMember = 0;                 // 部材番号の現在値
        POSITION pos;
        PROCESS_INFO *pInfo;
        if(m_listInfo.IsEmpty())  return;
        nProcess = m_listInfo.GetCount();
        nPart = (int)(m_nMem / nProcess) -1;
        nRemainder = m_nMem % nProcess;
```

```
        // 先頭のプロセスは余りを負担する
        for(pos = m_listInfo.GetHeadPosition(); pos != NULL; m_listInfo.GetNext(pos))
        {
                pInfo = (PROCESS_INFO*)m_listInfo.GetAt(pos);
                pInfo->m_nRange[0] = nMember + 1;
                if(pos == m_listInfo.GetHeadPosition())
                        nMember = pInfo->m_nRange[0] + nPart + nRemainder;
                else
                        nMember = pInfo->m_nRange[0] + nPart;
                pInfo->m_nRange[1] = nMember;
        }
        // リストを更新
        UpdateList();
}
```

3.5 マスター側の動的ソルバーにおける並列処理

本節では、マスター並列処理用の主サブルーチン Ver.1.10 について説明する。このサブルーチンの全ては付録に収められている。長いプログラムではあるが、ざーと目を通していただきたい。先に SPACE 動的解析編を読んでおけば全体を容易に理解でき、また違いが分かると思う。

ここでは、マスター用主サブルーチンの骨格を取り出し、特に並列用に変更した部分について説明する。このサブルーチンは SPACE の動的解析ソルバーsubmain_dynamic_a()を並列用に変更したもであり、変更箇所は、以下のようにまとめられている。

付録には、スレーブ制御部分が改良された Ver.1.11 が収録されている。

１） 分散並列型システム構築用
- MPI_Comm_size()
- MPI_Comm_rank()

２） コントロールデータ送信のための情報収集
- dyctl1_pa()
- doutcl_pa()
- damctl_pa()

３） データ送受信用
- send_ctlset()
- Send_structure()
- Send_imperfection()
- Send_Fiber()
- Send_RO_data()
- Send_damp()
- Convert_node_inf_vpp()
- recv_ld_repeat()
- Send_Results_acc()
- bcast_ite()
- recv_force_vpp()
- bcast_ite()
- recv_max_stress()

４） 並列処理時間計算用
　　MPI_WTime()
５） 解析用データ送信のための情報収集
　　Get_structure_pa()
　　Get_imperfection_pa()
　　Fiber_input_pa()
　　RO_data_input_pa()
６） 既存ソフトの変更
　　Get_pointforce_ld_pa()
　　Add_damp2_ld_pa()
　　Add_damp3_ld_pa()
　　Add_stiff1_ld_pa()
　　Add_stiff2_ld_pa()
　　Add_tan_stiff_ld_pa()
　　Add_fdd_ld_pa()
　　Cal_stress_pa ()
　　Check_stress_pa()
　　Check_Maxwell_stress_pa()
　　Get_max_stress_pa()
　　Get_nonlinear_stiff_pa()

上で分類した各サブルーチンの役割について説明しよう。1）は、マスター側が並列処理を行うために、SPACE システムから分担する部材番号やランクなどを取得するサブルーチンであり、これらについては、次章で説明する。2）は、制御データやコントロール情報をスレーブに送信するために、データをバッファ領域にパックするサブルーチン類である。ここでのプログラムは、単一 CPU プログラムを変更して用いている。3）は、スレーブとの送受信用サブルーチンと送信データをパックするためのテーブル作成サブルーチンである。これらのサブルーチンは、次章で詳細に説明する。

4）は、処理時間を計測するサブルーチンである。5）は、解析用データ送信のための情報収集用サブルーチンであり、既存の SPACE サブルーチンにデータをパックするためのコードを追加して用いている。内容については、次章を参照されたい。6）は、マスター側が分担する部材について処理を行うように、既存のプログラムコードの一部を変更して用いている。その内容は、既存のプログラムとほとんど同じである。

次に、前節で示したマスター側の主プログラムの骨格を取り出して、新たに並列用に変更した部分を中心にその内容を説明する。他の部分の説明は「マニュアル動的解析編」を参照されたい。

```
C------------------------------------------------------------------
C     ●   SUBROUTINE /submain_dynamic_a
C------------------------------------
```

```
C          Parallel version for Master
C      ------------------------------
C      ●    動的解析マスター用主プログラム（反復解法）Ver.1.10
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
c        マスター並列処理用　メインプログラム
c          マスターとモニターの ID は　 0
c          スレーブは                   1    以降連続番号で定義
c        基本計画
c            部材に関する動的領域はマスターは全部材、スレーブは担当部材のみ確保する。
c            要素及び節点データに関しては全てマスターと同じとする
c            自由度に関するデータはマスターは全ての領域、
c            マスターは担当部材が必要とする領域のみ使用する
c            スレーブが必要とする情報は、マスターより転送する
c
c  マスター側がスレーブに送受信するデータブロックは以下のようである。
c      Ⅰ。予備計算
c            1. コントロール情報送信（全スレーブ同じ情報）
c            2. 構造データ送信（全スレーブ同じ情報）
c            3. 初期不整データ送信（全スレーブ同じ情報）注１
c            4. ファイバーデータ送信（全スレーブ同じ情報）注１
c            5. RO モデル用データ送信（全スレーブ同じ情報）注１
c            6. 減衰用データ送信（全スレーブ同じ情報）注１
c            7. 初期変位送信（全スレーブ同じ情報：現在使用不可）
c            8. 初期応力送信（全スレーブ同じ情報：現在使用不可）
c               注１スレーブ側が必要としなくても、全スレーブに転送する。
c      Ⅱ。動的解析
c            1. 右辺項を受信（スレーブによって受信数が異なる）
c            2. 計算結果加速度データ送信（スレーブによって送信数が異なる）
c            3. 未収束コード送信（全スレーブ同じ情報）
c            4. 収束コード送信（全スレーブ同じ情報）
c            5. 部材両端の節点力を受信（スレーブによって受信数が異なる）
c      Ⅲ。後処理
c            1. 最大応力等をスレーブからの受信（スレーブによって受信数が異なる）
c
c  スレーブで送受信数が異なるデータはスレーブ転送用テーブルを用いて、
c  送信する場合は圧縮し、受信する場合は展開する。
c            1. スレーブ転送用テーブル作成　Convert_node_inf_vpp( ) Create_table( )
c            2. 送信圧縮　Dt_from_M_to_S( )
c            3. 受信展開　Dt_from_S_to_M( )
c
c    スレーブ制御コード
c            0:  解析開始　正常終了の場合は最後は制御コードなし
c            1:  発散・途中終了
c            2:  エラーは発生、途中終了
c            3:  未収束

c
c-----------------------------------------------------------------------
c        ** 解析番号と担当部材を GUI モニターから取得
cc
c          Model_No.1 = 1              !  幾何学非線形型有限要素弾性モデル
c          Model_No.2 = 2              !  3 次元せん断弾塑性モデル
c          Model_No.3 = 3              !  3 次元軸力弾塑性モデル
```

```
c        Model_No.4 = 4              !  3次元ケーブル弾塑性モデル
c        Model_No.5 = 5              !  3次元免振モデル（MSS モデル）
c        Model_No.6 = 6              !  3次元制震 Maxwell モデル
c        Model_No.7 = 7              !  3次元弾塑性バネモデル(*)
c
c
c        Model_No.11 = 11            !  両端ファイバーモデル
c        Model_No.12 = 12            !  両端、中央ファイバーモデル
c        Model_No.13 = 18            !  両端ピンで中央ファイバーモデル
c        Model_No.15 = 15            !  ファイバーモデル
c
c        Model_No.13 = 13            !  両端 MS モデル
c        Model_No.14 = 14            !  両端、中央 MS モデル
c
c        Model_No.31 = 16            !  両端アナロジーモデル
c        Model_No.32 = 17            !  両端、中央アナロジーモデル
c        Model_No.32 = 19            !  両端ピン、中央アナロジーモデル

c        Model_No.51 = 51            !  3次元プレテンション動作モデル
c
c        Model_No.50 = 1001          !  DLL 有限要素弾塑性モデル
c
c-------------------------------------------------------------------
c
c      ファイバー履歴タイプ
c      nm_type: 1     バイリニア型
c        No.1 = 1                    !  対称バイリニア
c        No.2 = 2                    !  対称トリリニア
c        No.3 = 3                    !  直線コンクリート型
c        No.4 = 4                    !  曲線線コンクリート型
c        No.5 = 5                    !  対称バイリニア型(移動＋等方効果用)
c        No.6 = 6                    !  対称トリリニア型（移動＋等方効果用）
c        No.7 = 7                    !  非対称バイリニア型
c        No.8 = 8                    !  非対称トリリニア型（移動＋等方効果用）
c
c        アナロジーモデル履歴タイプ
c              11     完全弾塑性型
c              12     等方硬化弾塑性型
c              13     移動硬化弾塑性型
c              14     等方硬化＋移動硬化弾塑性型
c
c        マルチスプリング履歴タイプ
c              21     武田モデル
c              22     等方硬化＋移動硬化トリリニア型
c

c-------------------------------------------------------------------
c
c      断面モデル
c        No.1                      !  円管
c        No.2                      !  角パイプ
c        No.3                      !  H　型
c        No.4                      !  長方形
c        No.5                      !  T　型
```

```
c        No.6                        !
c        No.7                        !
c
c-----------------------------------------------------------------------
c
c     部材モデルの階層構造は、以下のようである。
c     部材ー＞要素モデルー＞断面モデルー＞履歴モデル
c     １．部材は、各部材固有のデータを保持する。
c     ２．要素モデルは、部材の内部を構成するデータを保持する。
c     ３．断面モデルは、部材の断面形状を等のデータを保持する。
c     ４．履歴モデルは、各要素の弾塑性状態を制御するデータを保持する。
c
c     部材モデルの組み込み手順
c     １．Member_s と Element_s 構造体を設定する。
c     ２．弾塑性解析を実行するために必要となる領域（構造体）を設定する。
c     ３．Cal_stiff_linear に線形剛性を組み込む
c     ４．Cal_stress に応力計算を組み込む
c     ５．Check_stress に弾塑性チェックを組み込む
c     ６．Get_nonlinear_stiff に非線形剛性を組み込む
c
c      解析種別  N_analysis :
c           7: 線形解析
c           8: 幾何学的非線形解析
c           9: 弾塑性解析
c          10: 幾何学的＋弾塑性解析
c           6: 線形座屈解析
c
c     部材別解析種別 : Member(i).nm_analysis
c          -1: 線形解析
c           0: 全体解析種別に従う
c           1: 部材座標系 x 方向弾性
c           2: 部材座標系 y 方向弾性
c           3: 部材座標系 z 方向弾性
c
c-----------------------------------------------------------------------
c
c submain_dynamic_a  引数
c               iend_code       :計算終了コード 0=継続  1:終了
c               icontrol        :計算コード     0:予備計算  1:動的解析 99:終了処理
c               ierr_dat        :エラーコード
c               T               :計算時刻
c               dt              :解析増分時間
c               n_step          :今回の解析ステップ数
c               ns_step         :解析開始ステップ（最初はゼロセットが必要）
c               n_iterate       :反復回数
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     ★         サブルーチン定義
c
c-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C      ●    SUBROUTINE /submain_dynamic_a
C-------------------------------------
```

```
C          Parallel version for Master
C      ------------------------------
C      ●    動的解析マスター用主プログラム（反復解法）Ver.2.10
C-----------------------------------------------------------------------
      subroutine submain_dynamic_a()

c-----------------------------------------------------------------------
c-------------------------------------------------★並列処理に必要な宣言
      use MPI_DEFINE                                                  ! 1
      use TIME_MODULE                          ! 時間計測用定義・変数
c-----------------------------------------------------------------------
      implicit real*8(A-H,O-Z)
c-------------------------------------------------★動的解析制限条件

c-----------------------------------------------------------------------
c
c     ★          システムからの制御情報を取得（以後実行文）
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
      if(icontrol .eq. 1  ) goto 9990    ! 解析処理へ
      if(icontrol .eq. 99 ) goto 9997    ! 終了処理へ
      if(icontrol .eq. 98 ) goto 9998    ! 途中終了処理へ
      open (damp_out,FILE='DOUTPUT')
c------- ★★  システムからのコントロール情報を取得
c-------------------------------------------------★MPI 上での自己の状態を得る
      ! 解析プロセスの総数を取得
      call MPI_Comm_size(MPI_COMM_WORLD, n_proc, ierr)                ! 2
      ! 自分のランクを取得
      call MPI_Comm_rank(MPI_COMM_WORLD, n_myRank, ierr)              ! 3
      write(damp_out,'(a,2i5)')'(n_proc, myRank) =', n_proc, n_myRank
      ALLOCATE (nm_parallel( 2, 0:n_proc))                            ! 4
c-------------------------------------------------★担当部材リストを取得
      do i=0, n_proc-1                                                ! 5
      call RequestMember(nm_parallel, i)
      enddo
      n_member1=nm_parallel(1, n_myRank)          ! 担当部材の先頭番号  ! 6
      n_member2=nm_parallel(2, n_myRank)          ! 担当部材の最終番号
      max_member=n_member2-n_member1+1            ! 担当部材数
      write(damp_out,'(a,i5,a,i5,a,i5)')'max_member=',max_member,
     +' n_member1=',n_member1,' n_member2=',n_member2
c-------------------------------------------------★システムからのコントロール情報を取得(ok)
      call sysnam()
c-------------------------------------------------★コントロールデータの内容を取得(ok)
      call ctlset()
c------- ★★解析制御情報をファイルから入力し、スレーブに転送するためバッファにセット
c-------------------------------------------------★解析制御情報をファイルから入力(vpp)
      ALLOCATE (ff_data(400),if_data(400),iif_dt(2))                  ! 7
      iif_dt(1) = 0     ! ff_data の最終場所                          ! 8
      iif_dt(2) = 8     ! if_data の最終場所
c-------------------------------------------------★動的解析ダイアログその１のデータを入力(ok)
      call dyctl1_pa(ierr,NINDIT,GINDIS,F1SEC,fs_st,fl_st,ifp_st,IST, ! 9
     +               JIKUZERO,G_JIKUZERO_ALPH,
     +               ff_data(iif_dt(1)+1),if_data(iif_dt(2)+1),iif_dt)
```

```
c-------------------------------------------------------★動的解析ダイアログその２のデータを入力(ok)
      call dyctl2_pa(ierr,NSTEP,F2SEC,DELT,IGRA,IBETA,BETA,GUMMA,XGAL,          ! 10
     *           NNTIME,EPSDSP,load_memb_mass,dt_M_filter,IT_ANALYS,
     +           ff_data(iif_dt(1)+1),if_data(iif_dt(2)+1),iif_dt)
c-------------------------------------------------------★解析結果の出力パラメータを入力(ok)
      call doutcl_pa(ierr,IWSTP,SOUTSC,DMAXCK,No_section,                       ! 11
     +              ff_data(iif_dt(1)+1),if_data(iif_dt(2)+1),iif_dt)
c-------------------------------------------------------★減衰ダイアログのデータを入力(ok)
      call damctl_pa(ierr,NREAD,ITYDP,NDMP,NDMP2,NHH,HH,QHH,                    ! 12
     +              ff_data(iif_dt(1)+1),if_data(iif_dt(2)+1),iif_dt)
c-------------------------------------------------------★制御情報の取得に失敗した場合はここで戻る
      if(ierr_dat .ne. 0 ) then
      DEALLOCATE (ff_data,if_data,iif_dt)                                       ! 13
      return
      endif
c-----------------------------------------------------------------------
c
c     ★         制御情報を構造体にセット
c
c-----------------------------------------------------------------------
      call Set_control()        !ok (in_disp,in_stres  未定義)                  ! 14
      call Set_model_type()
      call Set_newmark()
      call Set_dynamic_load()
c-----------------------------------------------------------------------
c
c     ★         構造データを予備入力し、構造用のパラメータを設定する(ok)
c
c-----------------------------------------------------------------------
      call Get_parameters()                                                     ! 15
c-------------------------------------------------------★制御情報をスレーブ
c--------- ★★  制御情報と構造用制御情報を送信
      call send_ctlset(Parameter_C,ff_data,if_data,iif_dt, N_analysis)          ! 16
      DEALLOCATE (ff_data,if_data,iif_dt)                                       ! 17
c-------------------------------------------------------★地震加速度を予備入力(ok)
      call Get_earth_load()
c-------------------------------------------------------★節点荷重履歴を予備入力(ok)
      call Get_point_load()
c-----------------------------------------------------------------------
c
c     ★         構造体の大きさを動的確保する（その１）
c
c-----------------------------------------------------------------------
      ALLOCATE (Max_disp(Parameter_C.n_point))
       .
       .
c-----------------------------------------------------------------------
c
c     ★         構造・荷重データを入力し、データの設定を行う
c
c-----------------------------------------------------------------------
c-------------------------------------------------------★基本構造データを入力(ok)
c--------- ★★  構造データの送信用バッファ領域を確保
c     バッファデータ仕様
```

```
c      buf()     １：座標 3
c                ２：局所座標 3
c                ３：要素 17
c                ４：部材 4
c     ibuf()     １：境界拘束条件 7
c                ２：要素 6
c                ３：部材  14
c
       id_buf=Parameter_C.n_point*6+Parameter_C.n_element*17+                  ! 18
     *        Parameter_C.n_member*4
        jd_buf=Parameter_C.n_point*7++Parameter_C.n_element*6+
     *        Parameter_C.n_member*14
       ALLOCATE (buf(id_buf+10),ibuf(jd_buf+10))                               ! 19
c--------- ★★  構造データの送信用バッファ領域を確保
       call Get_structure_pa(Point,Member,Element,Parameter_C,                 ! 20
     *                       Model_type,ierr,buf,ibuf,id_buf,jd_buf)
c--------- ★★  スレーブ、モニターに構造データを送信
       call Send_structure(buf,ibuf,id_buf,jd_buf)                             ! 21
       DEALLOCATE (buf,ibuf)                                                   ! 22
        .
        .
c-----------------------------------------------------------------------
c
c     ★         計算用構造体の大きさを動的確保する（その２）
c
c-----------------------------------------------------------------------
       N=Model_type.n_m_damp
       if(N.ne.0)then
       ALLOCATE (ac_member(12,12,N))
       endif
        .
        .
c-----------------------------------------------------------------------
c
c     ★         構造・荷重データを入力し、データの設定を行う（その２）
c
c-----------------------------------------------------------------------
c-------------------------------------------------★地震荷重を入力(ok)
       call Get_earth_load()
c-------------------------------------------------★節点荷重分布を入力(ok)
       call Get_point_loadf()
c-----------------------------------★節点荷重時刻歴を入力およびセット
       call Get_point_load()
c-------------------------------------------------★初期不整データを入力(ok)
       if(Control.init_imperfection .ne. 0) then
       call Get_imperfection_pa()                                              ! 23
       id_buf=npoint_imp*4+1                                                   ! 24
       ALLOCATE (buf(id_buf+10))                                               ! 25
       call Get_imperfection_pa()                                              ! 26
c--------- ★★  スレーブに初期不整データを転送
       call Send_imperfection(buf,ii1_buf,npoint_imp)                          ! 27
       DEALLOCATE (buf)                                                        ! 28
       endif
c-------------------------------------------------★システム内要素データを入力
```

```
c     ★          ファイバーデータ
c-------------------------------------★ファイバーデータの予備入力(ok)
      call Fiber_input_pa()                                          ! 29
c-------- ★★スレーブにファイバー用バッファをゼロセット
      ALLOCATE (buf(id_buf+10),ibuf(jd_buf+10))                      ! 30
      do i=1,id_buf+10                                               ! 31
      buf(i)=0.
      enddo
      do i=1,jd_buf+10
      ibuf(i)=0
      enddo
c-------------------------------------★ファイバーモデル領域セット
      n= Model_type.nm_div_fmodel          !要素モデル内のサブ要素の数
      if(n.ne.0) then
      ALLOCATE (M_model_fiber(n))
      endif
      ・
      ・
c-------------------------------------★ファイバーモデルデータ入力
      call Fiber_input_pa(1,・)                                      ! 32
c-------- ★★スレーブにファイバーデータを転送
      call Send_Fiber(id_buf,jd_buf ,buf,ibuf)                       ! 33
      DEALLOCATE (buf,ibuf)                                          ! 34
      endif
c-------------------------------------★修正 RO モデル領域セット
      n=Model_type.n_m_ro_model
      if(n.ne.0) then
      call RO_data_input_pa()                                        ! 35
      ALLOCATE (buf(id_buf+10))                                      ! 36
      call RO_data_input_pa()                                        ! 37
c-------- ★★スレーブに RO モデル用データを転送
      call Send_RO_data(id_buf,jd_buf,buf)                           ! 38
      DEALLOCATE (buf)                                               ! 39
      endif
c-----------------------------------------------★質量データを入力(ok)
      call Get_mass()
c-----------------------------------------------★レーリー減衰を入力(ok)
      call Get_damp(Newmark_P,ierr)
c-------- ★★スレーブに減衰用データを転送
      call Send_damp(Newmark_P,ierr)                                 ! 40
c-----------------------------------------------------------------
c
c     ★          動的解析のための予備計算を行う
c
c-----------------------------------------------------------------
c-------- ★★  Time Check Codes
      call MPI_WTime(TIME_PRE, nStart)
c-----------------------------------------------★部材長さ計算(ok)
      call Cal_member_length()
c-----------------------------------------------★モデルの初期設定
      call Set_initial_data()
c-----------------------------------------------★座標変換行列計算
      call Get_rotate_all)
c-----------------------------------------------★局所座標変換行列計算
```

```
      call Get_rot_local()
c-----------------------------------------------------★釣合座標系標変換行列計算
      call Get_rotate()
c-----------------------------------------------------★節点拘束表の作成
c                                                     未知数等をセット
      call Set_restraint_point()
c-------------------------------------------------------------------------
c
c     ★         配列の大きさを動的確保する（その３）
c
c-------------------------------------------------------------------------
      N=Parameter_C.n_unknown
c      ALLOCATE (test_vector(N))
      if(n_proc.ge.2)then                                              ! 41
      ALLOCATE (Ms_table(N,n_proc-1),Ms_free(n_proc-1))
       endif
      ALLOCATE (
     *     disp_point(N),vel_point(N),acc_point(N),
     *     est_disp_point(N),est_vel_point(N),
     *     est_ddisp_point(N)
     *         )
       .
       .
c-----------------------------------------------------★部材両端の拘束表作成(ok)
      call Set_restraint_member()
c-----------------------------------------------------★スカイライン変換表作成(ok)
c-----------------------------------------------------★スカイライン行列の領域数等をセット
      call Cal_table_sky()
c-------------------------------------------------------------------------
c
c     ★         配列の大きさ（スカイライン用）を動的確保する（その４）
c
c-------------------------------------------------------------------------
c-----------------------------------------------------★ファイバー用履歴要素
      n= Model_type.n_m_bilinear          !  バイリニアの履歴要の数
      if(n.ne.0) then
      ALLOCATE (Bilinear_work(n))
      endif
       .
       .
c-----------------------------------------------★部材両端中央応力、力のゼロセット(ok)
      call  Set_pointforce_zero()
c-----------------------------------------------★部材応力のゼロセット(ok)
      call  Set_stress_zero()
c-----------------------------------------------------★MSS モデルの初期設定
      n=Model_type.n_m_model(5)
      if(n.ne.0) then
      call Cal_MSS_dat()
      endif
c-----------------------------------------------------★平面問題における部材の拘束方向チェック(ok)
      call Check_R_direction()
c-----------------------------------------------------★部材の線形剛性計算(ok)
      call Cal_stiff_linear()
c-----------------------------------------------------★剛性の釣合座標系への変換(ok)
```

```
      call Rotate_stiffness()
c-----------------------------------------------------★部材の減衰行列計算(ok)
      if(Parameter_C.nc_member .ne. 0) then
      call Cal_damp_linear()
 c----------------------------------------------------★部材減衰行列の釣合座標系への変換(ok)
      call Rotate_damp()
      end if
c-----------------------------------------------------★節点集中質量セット(ok)
      call Set_mass()
c-----------------------------------------------------★節点荷重セット(ok)
      call Set_point_load()
c-----------------------------------------------------★接線剛性のコピー(ok)
      call Initset_nonlin_stiff()
c-----------------------------------------------------★ベクトルのゼロセット
c-------------------------------------------★増分前の変位、速度、加速度(ok)
      call Set_zero_v()
c-------------------------------------------★最大変位等のゼロセット(ok)
      call Clear_max_disp()
c-------------------------------------------★最大、最小応力等のゼロセット(ok)
      call Clear_max_stress()
c-----------------------------------------------------------------------
c
c     ★         静的解析結果等を取り込む
c               （初期変位、初期応力を入力）
c
c-----------------------------------------------------------------------
c-----------------------------------------------------★初期変位を入力
c       call Get_init_disp(Point,Member,Parameter_C,ierr)
      close(nfix)
c-------- ★★スレーブに初期変位を転送
c     call Send_init_disp(Point,Member,Parameter_C,ierr,MPP_Ana_Group)
      endif
c-----------------------------------------------------★初期応力を入力
      if(Control.init_stress.ne.0) then
c      call Get_init_stress(Point,Member,Parameter_C,ierr)
c     write(damp_out,*) ' Get_init_stress Ok'
c-------- ★★スレーブに初期応力を転送
c     call Send_init_stress(Point,Member,Parameter_C,ierr,MPP_Ana_Group)
      endif
c-----------------------------------------------------------------------
c
c     ★         解析結果を出力するファイル群をオープンする(ok)
c
c-----------------------------------------------------------------------
      call flcheck()
c-----------------------------------------------------★出力指定した断面がファイバー要素
c-----------------------------------------------------★かどうかチェック(ok)
      call out_section_check()
c-----------------------------------------------------★ファイル名一覧出力
      do i=1,16
      write(damp_out,'(i4,3i6)') i,ifl(i),iflz(i),ifly(i)
      enddo
c-----------------------------------------------------------------------
c
```

```
c     ★         描画用データのセット
c
c-------------------------------------------------------------------------------
c-----------------------------------------★変位
      if(i_read_disp .ne. 0) then
c-------- ★★モニターの描画用データを作成
      call Set_preset_disp()
      endif
c-------------------------------------------------------------------------------
c
c     ★         解析ステップをセットする
c
c-------------------------------------------------------------------------------
      call MPI_WTime(TIME_ALL, nStart)                  ! 解析時間を計る
c-----------------------------------------------------★解析手法のセット
      N_implicit_method = 1  !陰解法は１回で反復法に戻る
      Iteration_method  = 1  !最初は反復法を使用
c-------- ★★加速度のスレーブ転送用テーブル作成
      if(n_proc.ge.2) then                                                  ! 42
      call Convert_node_inf_vpp(Parameter_C.n_unknown,Parameter_C,
     *      Member,Point,Ms_table,Ms_free,n_proc,nm_parallel)
      do i=1, n_proc-1
      write(damp_out,'(a,i3,a,i10)') 'ms_free(',i,')=',Ms_free(i)
      end do
      endif
c-------------------------------------------------------------------------------
c-----------------------------------------------★予備計算時間のセット
c-------- ★★  Time Check Codes
      call MPI_WTime(TIME_PRE, dTime_Pre)
      All_Time_Pre = All_Time_Pre + dTime_Pre
      stimes=0
c-------------------------------------------------------------------------------
c
c     ★         予備計算はここで終了（GUIに戻る）
c
c-------------------------------------------------------------------------------
      return
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
c
c     ★         動的解析開始
c
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
      i_print = 0
 9990 continue
      do 9999 istep=ns_step,ns_step+n_step -  1
c-----------------------------------------------★解析開始コードをスレーブに送る
c-------- ★★スレーブに解析開始コードを転送
c           スレーブ制御コード 0:  解析開始　正常終了の場合は最後は制御コードなし
c                              1:  発散・途中終了
c                              2:  エラーは発生、途中終了
c                              3:  未収束
            if ( n_proc .ge. 2)then
```

```
              ite_end = 0
              call bcast_ite(ite_end)                                    ! 43
              endif
c-------- ★★  Time Check Codes
c       write(damp_out,*) "動的解析"
        call MPI_WTime(TIME_STEP, nStart)
c-------------------------------------------------------------------------------
c
c     ★         第一と第二ステップの最初にスカイライン行列を作成、分解する
c
c-------------------------------------------------------------------------------
c-----------------------------------------------★左辺係数行列の計算(ok)
c-------------------------------------------★ステップ番号のセット(ok)
        if(istep.eq.1.or.istep.eq.Newmark_P.n2_step) then
c-------- ★★  Time Check Codes
        call MPI_WTime(TIME_K, nStart)
c-------------------------------------------★スカイライン行列のゼロセット(ok)
        call Set_sky_zero(gskym,n_skyline)
c-------------------------------------------★集中質量系の行列への組み込み
c-------------------------------------------★レーリー減衰を含む
        call Build_sky_mm()
c-------------------------------------------★部材の整合質量系の行列への組み込み(ok)
c-------------------------------------------★レーリー減衰を含む
c-----------------------------------★部材の整合質量行列計算(ok*)
        if(Dynamic_load.load_mass .ne. 0) then
        call Cal_mass_linear()
c-----------------------------------★整合質量の釣合座標系への変換(ok*)
        call Rotate_mass()
c-----------------------------------★整合質量系の組み込み(ok*)
        call Build_sky_m()
        endif
c-------------------------------------------★部材減衰系の組み込み(ok)
        if(Parameter_C.nc_member .ne. 0) then
        call Build_sky_c()
        endif
c-------------------------------------------★線形剛性の組み込み(ok)
c-------------------------------------------★レーリー減衰を含む
        call Build_sky_k()
c-----------------------------------------------★行列の LDU 分解(ok)
        call decomp_sky()
c-----------------------------------------------★解析時間のセット
c-------- ★★  Time Check Codes
        call MPI_WTime(TIME_K, dTime_K)
        All_Time_K = All_Time_K + dTime_K       !  係数行列作成
        endif
c-------------------------------------------------------------------------------
c
c     ★         動的解析の開始
c
c-------------------------------------------------------------------------------
        if(Control.type_analysis .ne. 7) goto 9981
c-------------------------------------------------------------------------------
c
c     ★         線形解析
```

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

```
c
c-------------------------------------------------------------------------
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_LD, nStart)
c-----------------------------------------------------★右辺の定数ベクトルゼロセット(ok)
      call Clear_vec()
c-----------------------------------------------------★地震加速度セット(ok)
      acc1=Get_Acc(T,1,acc_earth,Dynamic_load,Newmark_P.f1_T)
      acc2=Get_Acc(T,2,acc_earth,Dynamic_load,Newmark_P.f1_T)
      acc3=Get_Acc(T,3,acc_earth,Dynamic_load,Newmark_P.f1_T)
c-----------------------------------------★集中質量に関する慣性項
      call Add_earth1_ld()
c-----------------------------------------★整合質量に関する慣性項
      call Add_earth2_ld()
c-----------------------------------------------------★節点荷重のセット(ok)
      p1=Get_Ps(T,1,fdd_point,Dynamic_load)
      p2=Get_Ps(T,2,fdd_point,Dynamic_load)
      p3=Get_Ps(T,3,fdd_point,Dynamic_load)
      call Add_point_ld()
c-----------------------------------------------------★線形減衰項計算(ok)
c-----------------------------------------★集中質量(ok)
      call Add_damp1_ld()
c-----------------------------------------★整合質量(ok)
      call Add_damp2_ld_pa()
c-----------------------------------------★部材減衰(ok)
      call Add_damp3_ld_pa()
c-----------------------------------------------------★線形剛性項計算(ok)
c                                                      レーリー減衰も含む
      call Add_stiff1_ld_pa()
c-----------------------------------------------------★Δｔ秒後の変位と速度を予測(ok)
      call Estimate_disp_vel()
c-----------------------------------------------------★Maxwell 型モデルの計算(ok)
      call Add_fdd_ld_pa()
c      write(damp_out,*) ' Add_fdd_ld ok'
c-----------------------------------------★解析時間のセット
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_LD, dTime_LD)
      All_Time_LD = All_Time_LD + dTime_LD   !  右辺項の足し込み
c--------- ★★スレーブより右辺項を受信
      if( n_proc .ge. 2)then
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_NET_LD, nStart)
c-----------------------------------------
      call recv_ld_repeat()
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_NET_LD, dTime_NET_LD)
      All_Time_NET_LD = All_Time_NET_LD + dTime_NET_LD    ! 右辺項の通信
c-----------------------------------------
      endif
c-----------------------------------------------------★線形方程式を解く(ok)
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_CAL, nStart)
c-------------------------
      call solve_sky()
```

```
c      write(damp_out,*) ' solve_sky ok'
c--------- ★★  Time Check Codes
       call MPI_WTime(TIME_CAL, dTime_CAL)
       All_Time_CAL = All_Time_CAL + dTime_CAL      !  振動方程式を解く
       call MPI_WTime(TIME_DISP_VEL, nStart)
c-------------------------------------------------★加速度をスレーブに送信(ok)
c--------- ★★スレーブに計算結果を転送
       if( n_proc .ge. 2)then
c--------- ★★  Time Check Codes
       call MPI_WTime(TIME_NET_ACC, nStart)
c-------------------------------------------------
c      write(damp_out, '(a, i5)') 'Send_Result_acc:istep = ', istep
       call Send_Results_acc()
c--------- ★★  Time Check Codes
       call MPI_WTime(TIME_NET_ACC, dTime_NET_ACC)
       All_Time_NET_ACC = All_Time_NET_ACC + dTime_NET_ACC
c-------------------------------------------------
       endif
c-------------------------------------------------★加速度よりβ法に基づき変位と速度を計算(ok)
       call Cal_disp_vel()
c--------- ★★  Time Check Codes
c-------------------------------------------------★解析時間のセット
       call MPI_WTime(TIME_DISP_VEL, dTime_DISP_VEL)
       All_Time_DISP_VEL = All_Time_DISP_VEL + dTime_DISP_VEL    ! 変位と加速度の計算,予測
       goto 9980
c-----------------------------------------------------------------------
c
c    ★         非線形解析
c
c-----------------------------------------------------------------------
 9981 continue
c       if(Iteration_method .eq. 2) goto 9982
c-----------------------------------------------------------------------
c
c    ★         動的解析（反復解法）
c
c-----------------------------------------------------------------------
c--------- ★★  Time Check Codes
       call MPI_WTime(TIME_NEWMARK, nStart)         !  Newmarkβ法の反復解法部分
       call MPI_WTime(TIME_LD, nStart)                 !  右辺項の作成
c-------------------------------------------------★右辺の定数ベクトルゼロセット(ok)
       call Clear_vec()
c          write(damp_out,*) ' Clear_vec ok'
c-------------------------------------------------★部材節点力のセット(ok)
       call Get_pointforce_ld_pa()                                          ! 44
c-------------------------------------------------★地震加速度セット(ok)
       acc1=Get_Acc(T,1,acc_earth,Dynamic_load,Newmark_P.f1_T)
       acc2=Get_Acc(T,2,acc_earth,Dynamic_load,Newmark_P.f1_T)
       acc3=Get_Acc(T,3,acc_earth,Dynamic_load,Newmark_P.f1_T)
c       write(damp_out,'(a,4f10.3,i4)') ' Get_Acc ok'
c-------------------------------------------★集中質量に関する慣性項
       call Add_earth1_ld()
c-------------------------------------------★整合質量に関する慣性項
       call Add_earth2_ld()
```

```
c-------------------------------------------------★節点荷重のセット(ok)
      p1=Get_Ps(T,1,fdd_point,Dynamic_load)
      p2=Get_Ps(T,2,fdd_point,Dynamic_load)
      p3=Get_Ps(T,3,fdd_point,Dynamic_load)
      call Add_point_ld()
c-------------------------------------------------★線形減衰項計算(ok)
c-----------------------------------------★集中質量(ok)
      call Add_damp1_ld()
c-----------------------------------------★整合質量(ok)
      call Add_damp2_ld_pa()                                    ! 45
c-----------------------------------------★部材減衰(ok)
      call Add_damp3_ld_pa()                                    ! 46
c-------------------------------------------------★線形剛性項計算(ok)
c                                                  レーリー減衰も含む
      call Add_stiff1_ld_pa()                                   ! 47
c-------------------------------------------------★Δｔ秒後の変位と速度を予測(ok)
      n_err_roop = 0
 9991 continue
      nx =0
      call Estimate_disp_vel()
c-------------------------------------------------★解析時間のセット
c-------- ★★  Time Check Codes
      call MPI_WTime(TIME_LD, dTime_LD)
      All_Time_LD = All_Time_LD + dTime_LD   !  右辺項の足し込み
c-----------------------------------------------------------------------
c
c    ★         反復計算開始
c
c-----------------------------------------------------------------------
      n_roop=Newmark_P.max_repeat
      do iroop=1,n_roop
      write(damp_out,*) ' 反復回数：',iroop
c-------- ★★  Time Check Codes
      call MPI_WTime(TIME_NONLINER, nStart)
c-------------------------------------------------★反復に関連する右辺ベクトルのゼロセット(ok)
      call Clear_vec(n_unknown,ld_point_repeat)
c          write(damp_out,*) ' Clear_vec ok'
c-------------------------------------------------★線形剛性に関するベクトル(ok)
      call Add_stiff2_ld_pa()                                   ! 48
c      write(damp_out,*) ' Add_stiff2_ld ok'
c-------------------------------------------------★接線剛性に関する増分ベクトル(ok)
      call Add_tan_stiff_ld_pa()                                ! 49
c      write(damp_out,*) ' Add_tan_stiff_ld ok'
cc------------------------------------------------★Maxwell 型モデルの計算(ok)
      call Add_fdd_ld_pa()                                      ! 50
c      write(damp_out,*) ' Add_fdd_ld ok'
c-------------------------------------------------★右辺項を受け取り、総和をとる（新帯）
      if (n_proc .ge. 2)then
c-------- ★★スレーブより右辺項を受信
c-------- ★★  Time Check Codes
      call MPI_WTime(TIME_NET_LD, nStart)
c-----------------------------------------------------
c      write(damp_out,*) ' enter recv_ld_repeat'
      call recv_ld_repeat()                                     ! 51
```

```
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_LD, dTime_NET_LD)
      All_Time_NET_LD     = All_Time_NET_LD + dTime_NET_LD! 右辺項通信
c――――――――――――――――――――――――――――――――――――――――
      endif
c――――――――――――――――――――――――――――――――――――――――★右辺２項の和を取る(ok)
      call add_vec()
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NONLINER, dTime_NONLINER)
      All_Time_NONLINER = All_Time_NONLINER + dTime_NONLINER    ! 非線形項の足し込み
      call MPI_WTime(TIME_CAL, nStart)
c――――――――――――――――――――――――――――――――――――――――★線形方程式を解く(ok)
      n_skyline=Parameter_C.n_skyline
      call solve_sky()
c――――――――――――――――――――――――――――――――――――――――★解析時間のセット
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_CAL, dTime_CAL)
      All_Time_CAL = All_Time_CAL + dTime_CAL      !  振動方程式を解く
      call MPI_WTime(TIME_DISP_VEL, nStart)
c――――――――――――――――――――――――――――――――――――――――★加速度をスレーブに送信(ok)
c―――――― ★★スレーブに計算結果を転送
      if( n_proc .ge. 2)then
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_ACC, nStart)
c――――――――――――――――――――――――――――――――――――――――
      call Send_Results_acc(n_unknown,result_acc_point,                    ! 52
     *         nm_parallel,n_proc, Ms_table,Ms_free)
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_ACC, dTime_NET_ACC)
      All_Time_NET_ACC = All_Time_NET_ACC + dTime_NET_ACC! 予測加速度通信
c――――――――――――――――――――――――――――――――――――――――
      endif
c――――――――――――――――――――――――――――――――――――――――★β法に基づき加速度より変位と速度を計算(ok)
      call Cal_disp_vel()
      call MPI_WTime(TIME_DISP_VEL, dTime_Time_DISP_VEL)
      All_Time_DISP_VEL = All_Time_DISP_VEL + dTime_Time_DISP_VEL
c――――――――――――――――――――――――――――――――――――――――★収束したかチェック(ok)
      if(ICheck_error(iroop,n_point,Point,n_unknown,result_disp_point,
     *     est_disp_point, Newmark_P) .eq. 0) then
c――――――――――――――――――――――――――――――――――――――――★収束コードをスレーブに送る
c―――――― ★★スレーブに収束コードを転送
      if ( n_proc .ge. 2)then
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_CONV, nStart)
c――――――――――――――――――――――――――――――――――――――――
      ite_end = 0
      call bcast_ite()                                                     ! 53
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_CONV, dTime_NET_CONV)
      All_Time_NET_CONV = All_Time_NET_CONV + dTime_NET_CONV    ! 収束コード通信
c――――――――――――――――――――――――――――――――――――――――――――――――――
      endif
      goto 9980
      else
```

新しいバージョン Ver.1.11 として、加速度をスレーブに転送するサブルーチンを収束チェックを行った後に回す。加速度データの最後尾にスレーブの制御コードを付け加える。詳細は第 4.4.7 節で述べる。

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

```
c―――――― ★★スレーブに未収束コードを転送
      if ( n_proc .ge. 2)then                                          ! 54
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_CONV, nStart)
c-----------------------------------------------------------------------
      ite_end=3
      if(iroop.eq.n_roop) ite_end=2     ! 収束計算失敗
      call bcast_ite(ite_end)
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NET_CONV, dTime_NET_CONV)
      All_Time_NET_CONV = All_Time_NET_CONV + dTime_NET_CONV     ! 収束コード通信
c-----------------------------------------------------------------------
      endif
      endif
c-----------------------------------------------------★次の増分値を予測(ok)
      nx=iroop
      call Estimate_disp_vel()
      end do
c-----------------------------------------------------------------------
c
c     ★         収束しなかった時の後処理
c
c-----------------------------------------------------------------------
      write(76,*) '  収束エラー発生、解析終了'
      n_iterate = 9999
      ierr_dat =30
      call err_outf(ieer_dat)
c-----------------------------------------------------------------------
c
c     ★         スレーブに終了処理を通知
c
c-----------------------------------------------------------------------
      return                                                           ! 55
 9982 continue

c-----------------------------------------------------------------------
c
c     ★         計算終了・後処理開始
c
c-----------------------------------------------------------------------
 9980 continue
c-----------------------------------------------★解析時間のセット
c―――――― ★★  Time Check Codes
      call MPI_WTime(TIME_NEWMARK, dTime_NEWMARK)
      All_Time_NEWMARK = All_Time_NEWMARK + dTime_NEWMARK！  Newmarkβ法　反復解法
c-----------------------------------------------★解析結果・増分変位をセット(ok)
      call Set_ddisp()
c-----------------------------------------------★変位、速度、加速度を出力
      vacc(1)=Get_Acc(T,1,acc_earth,Dynamic_load,Newmark_P.f1_T)
      vacc(2)=Get_Acc(T,2,acc_earth,Dynamic_load,Newmark_P.f1_T)
      vacc(3)=Get_Acc(T,3,acc_earth,Dynamic_load,Newmark_P.f1_T)
      call Out_disp_vel_acc()
c-----------------------------------------------★変位、速度、加速度の最大値セット(ok)
      call Get_max_disp()
```

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

```
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_STRESS, nStart)
c-------------------------------------------------------------------------
      call Cal_stress_pa ()                                          ! 56
c-------------------------------------------------------★計算時間のセット
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_STRESS, dTime_STRESS)
      All_Time_STRESS = All_Time_STRESS + dTime_STRESS    ! 応力計算
      call MPI_WTime(TIME_F_STRESS, nStart) ! ファイバー応力の計算開始
c-------------------------------------------------------------------------
c
c    ★         部材の弾塑性状態をチェック
c
c-------------------------------------------------------------------------
c-------------------------------------------------------★部材塑性をチェック
c-------------------------------------------------------★部材両端の節点力計算（ok)
c     write(damp_out,*)"部材塑性をチェック"
c-------------------------------------------------------★ファイバー応力セット
      call Check_stress_pa()                                         ! 57
c-------------------------------------------------------★Maxwell 型モデルの非線形性チェック(ok)
      call Check_Maxwell_stress_pa()                                 ! 58
c--------- ★★部材両端の節点力を受け取る(新帯)
      if(i_print.eq.0) then                    ! ファイル出力のときのみデータを取得
      if( n_proc .ge. 2)then
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_NET_FORCE, nStart)
c-----------------------------------------------------
      call recv_force_vpp()                                          ! 59
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_NET_FORCE, dTime_NET_FORCE)
      All_Time_NET_FORCE = All_Time_NET_FORCE + dTime_NET_FORCE ! 部材端節点力通信
c-----------------------------------------------------
      endif
      endif
c-------------------------------------------------------★計算時間のセット
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_F_STRESS, dTime_F_STRESS)
      All_Time_F_STRESS = All_Time_F_STRESS + dTime_F_STRESS    ! ファイバー応力計算
c-------------------------------------------------------★部材応力を出力
      call Out_stress()
c       write(damp_out,*) ' Out_stress ok'
c-------------------------------------------------------★部材応力を出力
      call Out_Fiber()
c       write(damp_out,*) ' Out_stress ok'
c-------------------------------------------------------★応力等の最大値セット
      call Get_max_stress_pa()                                       ! 60
c      write(damp_out,*) ' Get_max_stress ok'

c-------------------------------------------------------★解析結果・変位、速度、加速度をセット(ok)
      call Set_disp_vel_acc()                    ! past_dacc_point に過去の加速度をセットする
c       write(damp_out,*) ' Set_disp_vel_acc ok'
c-------------------------------------------------------★不釣合い力（反力と荷重）の出力(ok)
      call Get_pointforce()
      call Out_pointforce()
```

```
c-------------------------------------------------★不釣合い力の計算
c-----------------------------------------★節点静的荷重項(ok)
      if(ifl(2).eq.1.and.i_print.eq.0) then
      p1=Get_Ps(T,1,fdd_point,Dynamic_load)
      p2=Get_Ps(T,2,fdd_point,Dynamic_load)
      p3=Get_Ps(T,3,fdd_point,Dynamic_load)
      call Add_pointforce()
c-----------------------------------------★集中質量に関する地震項
      vacc(1)=Get_Acc(T,1,acc_earth,Dynamic_load,Newmark_P.f1_T)
      vacc(2)=Get_Acc(T,2,acc_earth,Dynamic_load,Newmark_P.f1_T)
      vacc(3)=Get_Acc(T,3,acc_earth,Dynamic_load,Newmark_P.f1_T)
      call Add_earth1_pointforce()
c-----------------------------------------★整合質量に関する地震項
      call Add_earth2_pointforce()
c-------------------------------------------------★慣性項と減衰項計算(ok)
c-----------------------------------------★集中質量の慣性項と減衰項(ok)
      call Add_damp1_pointforce()
c-----------------------------------------★整合質量の慣性項と減衰項(ok)
      call Add_damp2_pointforce()
c-----------------------------------------★部材減衰の減衰項(ok)
      call Add_damp3_pointforce()
c-----------------------------------------★線形剛性のレーリー減衰項
      call Add_stiff1_pointforce()
c-----------------------------------------★不釣合い力の出力(ok)
      call Out_pointforce()
      endif
c-------------------------------------------------★終了か？（ステップと最大値チェック）
      if(istep .ge. Newmark_P.nn_step .or.
     *    d_max_v .gt.Control.collapse_maxdisp ) goto 9998
c-------------------------------------------------★接線剛性の計算(ok)
c     write( damp_out,*)"接線剛性の計算"
c------- ★★  Time Check Codes
      call MPI_WTime(TIME_KT, nStart)        ! 接線剛性の計算開始
c-------------------------------------------------
      call Get_nonlinear_stiff_pa()                                    ! 61
c-----------------------------------------★解析時間のセット
c------- ★★  Time Check Codes
      call MPI_WTime(TIME_KT, dTimeKT)
      All_Time_KT = All_Time_KT + dTimeKT    !  接線剛性作成
c-----------------------------------------★解析時間のセット
      call MPI_WTime(TIME_STEP, dTime_STEP)
      All_Time_STEP = All_Time_STEP + dTime_STEP   ! ステップ毎の計算時間
 9999 continue
c-----------------------------------------------------------------------
c
c    ★         時間増分更新
c
c-----------------------------------------------------------------------
      iend_code = 0
      ns_step=ns_step + n_step
c-----------------------------------------------------------------------
c
c    ★         描画用データのセット
c
```

```
c-------------------------------------------------------------------------
c-----------------------------------------★変位
      if(i_read_disp .ne. 0) then
c--------- ★★描画用データのセット
      call Set_preset_disp()
      endif
c-----------------------------------------★応力
      if(i_read_spring .ne. 0) then
      call Set_preset_spring()
c     write(damp_out,*) ' Set_preset_spring ok'
      endif
c
c-------------------------------------------------------------------------

      return
c-------------------------------------------------------------------------
c
c     ★         応答解析終了処理
c
c-------------------------------------------------------------------------
 9998 continue
      ite_end=1
c--------- ★★スレーブに解析終了コードを転送
      if ( n_proc .ge. 2)then                                          ! 62
      call bcast_ite(ite_end)
      endif
c-------------------------------------------------------------------------
c
c     ★         描画用データのセット
c
c-------------------------------------------------------------------------
c-----------------------------------------★変位
      if(i_read_disp .ne. 0) then
c--------- ★★描画用データのセット
      call Set_preset_disp()
      endif
c-----------------------------------------★応力
      if(i_read_spring .ne. 0) then
      call Set_preset_spring()
      endif
9997 continue
c-------------------------------------------------★最大変位、速度、加速度の出力
      call Out_max_disp()
c--------- ★★  最大応力等をスレーブからの受信
      if( n_proc .ge. 2)then                                           ! 63
      call recv_max_stress()
      endif
c-------------------------------------------------★最大応力等の出力
      call Out_max_stress()
c-------------------------------------------------★解析時間のセット
c--------- ★★  Time Check Codes
      call MPI_WTime(TIME_STEP, dTime_STEP)
      All_Time_STEP = All_Time_STEP + dTime_STEP   ! ステップ毎の計算時間
      call MPI_WTime(TIME_ALL, All_Time_ALL)             ! 総解析時間
```

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

```
c-------------------------------------------------★解析時間の出力
      write(76,'(a)')
     +             '解析時間の出力          :      秒'
      write(76,'(a,e16.5)')
     +             '予備計算                :',All_Time_PRE / 1000000
      write(76,'(a,e16.5)')
     +             '係数行列作成            :',All_Time_K / 1000000
      write(76,'(a,e16.5)')
     +             'Newmarkβ法              :',All_Time_NEWMARK / 1000000
      write(76,'(a,e16.5)')
     +             '-右辺項作成             :',All_Time_LD / 1000000
      write(76,'(a,e16.5)')
     +             '-非線形項作成           :',All_Time_NONLINER / 1000000
      write(76,'(a,e16.5)')
     +             '--通信時間(非線形項)    :',All_Time_NET_LD / 1000000
      write(76,'(a,e16.5)')
     +             '-振動方程式を解く       :',All_Time_CAL / 1000000
      write(76,'(a,e16.5)')
     +             '--通信時間(予測加速度):',All_Time_NET_ACC / 1000000
      write(76,'(a,e16.5)')
     +             '-変位、速度の計算       :',All_Time_DISP_VEL / 1000000
      write(76,'(a,e16.5)')
     +             '--通信時間(収束コード):',All_Time_NET_CONV / 1000000
      write(76,'(a,e16.5)')
     +             '応力の計算              :',All_Time_STRESS / 1000000
      write(76,'(a,e16.5)')
     +             '--通信時間(応力)        :',All_Time_NET_FORCE / 1000000
      write(76,'(a,e16.5)')
     +             'ファイバー応力の計算    :',All_Time_F_STRESS / 1000000
      write(76,'(a,e16.5)')
     +             '接線剛性作成            :',All_Time_KT / 1000000
      write(76,'(a,e16.5)')
     +             '総解析時間              :',All_Time_STEP / 1000000
      write(76,'(a,e16.5)')
     +             '総時間                  :',All_Time_ALL / 1000000
c-------------------------------------------------★ファイルのクローズ
      do i=1,16
       if(ifl(i).eq.1) close(iflz(i))
       enddo
c-------------------------------------------------★ファイルのタイムスタンプ
      call ctlset()
      call fltime(ifl,ifly,N_analysis,iflout)
      if(iflout.eq.1) call ctlset(ihan,FNX_file,TITLEX,IDFILE,NFILE)
c-----------------------------------------------------------------------
c
c     ★          動的配列の解放
c
c-----------------------------------------------------------------------
c----------------★          配列の動的領域を解放する（その１）
      if(M_alloc(1).eq.1)then
      DEALLOCATE (Point,Element,Member,Max_disp,Max_stress)        !ok
      DEALLOCATE (fll_static_point,am_point,fll_force_point)        !ok
      DEALLOCATE (am_member,rot_memb_t,rot_memb,ak_linear,ak_nonlinear)        !ok
       N=Parameter_C.n_local_coord
```

```
      if(N.ne.0) DEALLOCATE (rot_local)       !ok
      endif
      .
      .
c-------------------------------------------------------------------------
c
c     ★         並列処理で使用した動的配列の解放
c
c-------------------------------------------------------------------------
      if(n_proc.ge.2) then                                              ! 64
      DEALLOCATE (nm_parallel)
      DEALLOCATE (pa_work_force)
      endif
c-----------------------------------------★計算終了コードセット
      iend_code = 1
      close (damp_out)
      return
c-------------------------------------------------------------------------
c
c     ★         解析終了
c
c-------------------------------------------------------------------------
      end
```

プログラムの内容をコードの右端のコメント番号に従って説明する。この主サブルーチンの基本的な解析の流れに関する解説は、マニュアル：動的解析編を参照されたい。ここでは、並列処理に関連する部分について説明する。ただし、各処理の時間計測を行う処理については説明を省く。

1. 変数のバイト数など、MPI 準拠に基づいた定数宣言を行っているファイル MPI_DEFINE を、use 文を用いてインクルードする。同じく、ファイル TIME_MODEULE もインクルードする。
2. SPACE システムより、分散並列型システムを構成するプロセス総数を取得する。
3. 同じく、分散並列システムより自分のランク n_myRank を取得する。ただし、マスターのランクは 0 である。
4. 各プロセスの分担部材番号を保持する配列 nm_parallel の動的領域を確保する。
5. 各プロセスが分担する部材番号をシステムより取得する。
6. 自分自身が分担する部材番号をセットする。n_member1 は、担当部材の先頭番号、n_member2 は担当部材の最終番号、max_member は、担当部材数を表す。
7. 制御情報をスレーブに送信するためのバッファ領域を動的確保する。

ff_data は実数領域、if_data は整数データ領域であり、ここでは、少し領域を大きく取っている。

8. 実数データの個数を数える変数は、iif_dt(1)であり、ここでは、その最終個数がセットされている。整数については iif_dt(2)であり、その最終個数として、8 がセットされている。整数データとして、１から８までは、構造データの制御データをセットするために領域を確保している。
9. 動的解析ダイアログその 1 に関するデータファイルを入力し、データを該当する変数に設定すると共に、バッファ領域にセットする。この時、そのバッファ領域のデータ個数は、このサブルーチン内で更新される。
10. 動的解析ダイアログその 2 に関するデータファイルを入力し、データを該当する変数に設定すると共に、バッファ領域にセットする。この時、そのバッファ領域のデータ個数は、このサブルーチン内で更新される。
11. 解析結果の出力パラメータに関するデータファイルを入力し、データを該当する変数に設定すると共に、バッファ領域にセットする。この時、そのバッファ領域のデータ個数は、このサブルーチン内で更新される。
12. 減衰ダイアログに関するデータファイルを入力し、データを該当する変数に設定すると共に、バッファ領域にセットする。この時、そのバッファ領域のデータ個数は、このサブルーチン内で更新される。
13. 制御データの入力・設定に失敗した場合、このサブルーチンより直ちに戻ることになるが、その前に、先に動的確保した領域を解放する。
14. 解析制御用データを入力したが、4 つのサブルーチンによって、構造体にそれらのデータをセットする。
15. 構造データを仮読みし、構造データ用の制御データを取得する。
16. この構造データ用の制御データを整数用のバッファ領域にセットした後、全スレーブにデータを送信する。
17. 解析制御用のバッファとして、動的確保した領域を解放する。
18. 構造データ用の制御データを利用して、構造データをパックするための領域の大きさを計算する。id_buf は実数データの個数、jd_buf は整数の個数である。
19. 構造データをスレーブに送信するための領域を、動的に確保する。
20. 構造データを入力し、該当する構造体などにデータをセットすると

共に、バッファ領域にデータをパックする。構造データは、全てパックし、スレーブには入力データは全て送信することになる。
21.パックした構造データを全スレーブに送信する。
22.構造データ用に動的確保した領域を解放する。
23.初期不整が存在する場合は、初期不整データを仮読みし、個数を取得する。
24.初期不整用のバッファ領域の大きさを計算する。
25.初期不整用のバッファ動的領域を確保する。
26.初期不整データを入力し、構造体にセットすると共に、バッファにデータをセットする。
27.初期不整データを全スレーブに送信する。
28.初期不整用の動的領域を解放する。
29.ファイバー断面を有する場合、ファイバーデータの１回目のデータ入力を行う。ファイバーデータをパックするためのファイバーデータの個数を数える。
30.上記で数えた個数を用いて、バッファ領域の動的確保を行う。
31.バッファ領域をゼロクリアする。
32. ファイバーデータの２回目のデータ入力を行い、バッファ領域にデータをパックする。
33.ファイバーデータを全スレーブに送信する。
34.動的領域を解放する。
35.修正 RO モデルがある場合は、データを仮読みする。送信用バッファの個数をセットする。
36.バッファ用の動的領域を確保する。
37.RO データを入力し、構造体にセットすると共に、バッファにデータをパックする。
38.パックしたデータを全スレーブに送信する。
39.バッファ用の動的領域を解放する。
40. 構造体 Newmark_P の中の減衰データを全スレーブに送信する。
41.プロセスが２以上、つまり、スレーブを利用して並列処理を実行する場合は、スレーブにデータ転送するためのテーブルを作成する。ここでは、そのテーブルの動的領域を確保する。
42. プロセスが２以上の場合、スレーブデータ転送用テーブルを作成する。

これまでは、準備計算におけるデータ送信が主であったが、ここから

は、動的解析を実行するためのデータの送受信が必要となる。さらに、マスター側で解析の流れを制御しているが、それに合わせてスレーブ側も動作させるために、制御用コードをスレーブ側に送信する必要がある。ここでは、これらの処理を中心に説明を行う。

43.動的解析を実行するための解析ループに入った直後、解析を進めるコードを全スレーブに送信する。
44.部材節点力を右辺項ベクトルに足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
45.整合質量によって生じるレーリー減衰項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
46. 部材減衰によって生じるレーリー減衰項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
47. 線形剛性によって生じるレーリー減衰項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
48.線形剛性に関する項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
49. 接線剛性に関する項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
50.Maxwell モデルに関する項を右辺項に足しこむ。ただし、マスター側で担当する部材について、この処理を行う。
51. プロセスが２以上の場合、各スレーブ計算した右辺項を受信し、その値をマスター側の右辺項に足しこむ。
52.プロセスが２以上の場合、連立方程式を解いて得た加速度ベクトルを、各スレーブに送信する。この際スレーブデータ転送用テーブルを用いて、スレーブごとに加速度ベクトルをパックして送信する。
53. プロセスが２以上の場合、全スレーブに反復計算収束コードを送信する。
54. プロセスが２以上の場合、全スレーブに反復計算未収束コードを送信する。
55.最大反復回数を超えて、収束しなかったためエラーメッセージを出力してこのサブルーチンを抜ける。SPACE の単一 CPU 用の動的ソルバーでは、ここで陰解法に切り替えて、反復計算が収束しなかった場合に備えているが、現在のバージョンでは、なにもしていない。
56.部材の応力を求める。ただし、マスター側で担当する部材について、ここの処理を行う。

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

新しいバージョン Ver.1.11 として、加速度をスレーブに転送するサブルーチンを収束チェックを行った後に回す。加速度データの最後尾にスレーブ制御コードを付け加える。
詳細は第 4.4.7 節で述べる。

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

57.部材の弾塑性チェックを行う。ただし、マスター側で担当する部材について、ここの処理を行う。
58.Maxwell モデルの弾塑性チェックを行う。ただし、マスター側で担当する部材について、ここの処理を行う。
59. プロセスが２以上の場合、部材の応力をファイルに出力するために、各スレーブで計算した部材の応力を受信する。
60.担当する部材の最大応力を求め、構造体にセットする。
61. プロセスが２以上の場合、解析終了コードを全スレーブに送信する。
62. 担当する部材の接線剛性行列を求め、配列にセットする。
63.動的解析が終了した後、プロセスが２以上の場合、各スレーブの担当部材の最大応力を受信する。
64.並列処理用の動的領域を解放する。

Ver.1.11 では、この制御コード送信用サブルーチンを取り除く

3.6 スレーブ側の動的ソルバーシステム

本節からは、スレーブ側のシステム並びに動的ソルバーについて説明する。スレーブ側の動的解析システムは、各 PC で全て同じであり、コンソールアプリケーションである。スレーブ側システムは、同じ PC 内のデーモンによって起動される。そのデーモンは、マスター側システムによってスレーブシステムを起動するように命令される。

スレーブシステムは非常に単純で、起動した後、2 つの仕事を行う。ひとつは､PC クラスターをマスターと共同で構築することであり､他は、動的ソルバーを起動させることである。

最初は MPI_Init 関数によってソケットを作成し、一つ前の ID を有するプロセスからプロセス情報を取得する。この情報を基に他のプロセスと接続する。MPI_Init 関数は、

```
MPI_Init(argc, argv)
argc は argv[]の個数を表し、
argv[0] : 実行ファイル名
argv[1] : プロセス ID
argv[2] : 自身の ID よりひとつ若い PC の ID
```

であり、これらの引数は同じ PC 内のデーモンによってデータが受け渡される。この処理については次章を参照されたい。

接続に成功すると､スレーブ用の動的ソルバーに起動をかける｡後は､主サブルーチン SUBMAIN_DYNAMIC_S()の中で、情報の送受信を行う。

```
//  ------------------------------------------------------------
//    ●   sf3Slave.cpp
//  ------------------------------------------------------------
//    ●    コンソール アプリケーション
//  ------------------------------------------------------------
#include "mpi_c.h"
#include <iostream.h>
//  ------------------------------------------------------------
//    ●    FORTRAN 用の C++の外部宣言(ok)
//  ------------------------------------------------------------
extern "C"{
      void __stdcall SUBMAIN_DYNAMIC_S(int*,int*,int*,int*);
}

int main(int argc, char* argv[])
{
      int nRetCode = 0;
      cout << "rank = " << argv[1] << endl;
//  ------------------------------------------------------------
//    ●    MPI の初期化
//  ------------------------------------------------------------
      if(!MPI_Init(argc, argv))
      {
```

```
            return 0;
        }
        cout << "Slave スタート" << endl;
        int nCalnum;              // 解析種別
        int nEndCode = 0;         // 解析終了コード
        int nControl = 0;         // 解析コード(0 = 予備計算)
        int nErrCode = 0;         // エラーコード
//      ------------------------------------------------------------
//      ●    解析開始
//      ------------------------------------------------------------
        SUBMAIN_DYNAMIC_S(&nCalnum, &nEndCode, &nControl, &nErrCode);
//      ------------------------------------------------------------
//      ●    解析終了：エラー処理
//      ------------------------------------------------------------
        if(nErrCode != 0)
        {
            cout << "エラーがありました。計算を終了します" << endl;
            MPI_Send(&nErrCode, 1, MPI_INT, ID_MASTER, MPI_ANY_TAG, MPI_COMM_WORLD);
            return -1;
        }

        return 0;
}
```

3.7 スレーブ側の主サブルーチンの骨格

本節では、スレーブ用主サブルーチンの骨格を取り出し、特に、並列用に変更した部分を示す。このサブルーチンの全ては付録で示す。このサブルーチンは SPACE の動的解析ソルバーsubmain_dynamic_S()を並列用に変更したもであり、変更箇所は、以下のようにまとめられる。

１） 分散並列型システム構築用
- MPI_Comm_size()
- MPI_Comm_rank()

２） コントロールデータ送信のための情報収集
- dyctl1_pa()
- doutcl_pa()
- damctl_pa()

３） データ送受信用
- send_ctlset()
- Send_structure()
- Send_imperfection()
- Send_Fiber()
- Send_RO_data()
- Send_damp()
- Convert_node_inf_vpp()
- recv_ld_repeat()
- Send_Results_acc()
- bcast_ite()

```
        recv_force_vpp()
        bcast_ite()
        recv_max_stress()
４） 並列処理時間計算用
        MPI_WTime()
５） 解析用データ送信のための情報収集
        Get_structure_pa()
        Get_imperfection_pa()
        Fiber_input_pa()
        RO_data_input_pa()
６） 既存ソフトの変更
        Get_pointforce_ld_pa()
        Add_damp2_ld_pa()
        Add_damp3_ld_pa()
        Add_stiff1_ld_pa()
        Add_stiff2_ld_pa()
        Add_tan_stiff_ld_pa()
        Add_fdd_ld_pa()
        Cal_stress_pa ()
        Check_stress_pa()
        Check_Maxwell_stress_pa()
        Get_max_stress_pa()
        Get_nonlinear_stiff_pa()
```

上記は、前節で示したマスター用の変更分類と同じである。1）は、マスター側が並列処理を行うために、SPACE システムから分担する部材番号やランクなどを取得するサブルーチンであり、これらについては、次章で説明する。2）は、制御データやコントロール情報をスレーブに送信するために、データをバッファ領域にパックするサブルーチン類である。ここでのプログラムは、単一 CPU プログラムを変更して用いている。3）は、スレーブとの送受信用サブルーチンと送信データをパックするためのテーブル作成サブルーチンである。これらのサブルーチンは、次章で詳細に説明する。

4）は、処理時間を計測するサブルーチンである。5）は、解析用データ送信のための情報収集用サブルーチンであり、既存の SPACE サブルーチンにデータをパックするためのコードを追加して用いている。内容については、次章を参照されたい。6）は、マスター側が分担する部材について処理を行うように、既存のプログラムコードの一部を変更して用いている。その内容は、既存のプログラムとほとんど同じである。

次に、スレーブ側の主プログラムの骨格を取り出して、新たに並列用に変更した部分を中心にその内容を説明する。他の部分については、「マニュアル：動的解析編」を参照されたい。

```
C-----------------------------------------------------------------------
C      ●    SUBROUTINE /submain_dynamic_S
C      ------------------------------------
C           Parallel version for Slave
C      ------------------------------------
C      ●    動的解析スレーブ用主プログラム（反復解法）Ver.1.10
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
c      スレーブ並列処理用　メインプログラム
c       基本計画
c          部材に関する動的領域は担当部材のみ確保する。
c          要素及び節点データに関しては全てマスターと同じとする
c          自由度に関するデータは担当部材が必要とする領域のみ使用する
c          必要な情報は、マスターがファイルより取得し、スレーブには転送する
c
c       ** 解析番号と担当部材をマスターから取得
c
c          １．予備計算　：　部材に関するデータのみ処理する
c          ２．反復計算　：　圧縮した加速度を受信する
c                            圧縮した右辺項を送信する
c          ３．応力計算　：　マスターが出力するための応力を送信する
c
c  スレーブ側がマスターに送受信するデータブロックは以下のようである。
c     Ⅰ。予備計算
c          1. コントロール情報受信（全スレーブ同じ情報）recv_ctlset( )
c          2. 構造データ受信（全スレーブ同じ情報）recv_structure( )
c          3. 初期不整データ受信（全スレーブ同じ情報）注１recv_imperfection( )
c          4. ファイバーデータ受信（全スレーブ同じ情報）注１recv_Fiber( )
c          5. RO モデル用データ受信（全スレーブ同じ情報）注１recv_RO_data( )
c          6. 減衰用データ受信（全スレーブ同じ情報）注１recv_damp( )
c          7. 初期変位受信（全スレーブ同じ情報：現在使用不可）recv_init_disp( )
c          8. 初期応力受信（全スレーブ同じ情報：現在使用不可）recv_init_stress( )
c             注１担当部材にこの部材モデルがなくても、解析モデルで使用している場合は
c                 送信されるため受信なければならない。
c     Ⅱ。動的解析
c          1. 右辺項を送信 send_ld_repeat( )
c          2. 計算結果加速度データ受信 recv_result_acc( )
c          3. 未収束・収束コード受信（全スレーブ同じ情報）bcast_ite( )
c          4. 部材両端の節点力を送信 send_force_vpp( )
c     Ⅲ。後処理
c          1. 最大応力等をスレーブからの送信 send_max_stress( )
c
c  スレーブでは、送受信数が異なる。マスターから送信されたデータは、そのまま使用する。
c  各スレーブでは、担当部材に従って、未知数番号を作り直す。
c
c          1. スレーブ用テーブル作成　Reset_ij_table( )注２
c             注２：節点変位を同一視する場合、その相手が担当部材に連結していない場合
c                   を考慮しなければならない。
c
c
c-----------------------------------------------------------------------
c
c        Model_No.1 = 1              !　幾何学非線形型有限要素弾性モデル
c        Model_No.2 = 2              !　3 次元せん断弾塑性モデル
```

```
c        Model_No.3 = 3              !  3次元軸力弾塑性モデル
c        Model_No.4 = 4              !  3次元ケーブル弾塑性モデル
c        Model_No.5 = 5              !  3次元免振モデル（MSS モデル）
c        Model_No.6 = 6              !  3次元制震 Maxwell モデル
c        Model_No.7 = 7              !  3次元弾塑性バネモデル(*)
c
c        Model_No.11 = 11            !  両端ファイバーモデル
c        Model_No.12 = 12            !  両端、中央ファイバーモデル
c        Model_No.13 = 18            !  両端ピンで中央ファイバーモデル
c        Model_No.15 = 15            !  ファイバーモデル
c
c        Model_No.13 = 21            !  両端 MS モデル
c        Model_No.14 = 22            !  両端、中央 MS モデル
c
c        Model_No.31 = 31            !  両端アナロジーモデル
c        Model_No.32 = 32            !  両端、中央アナロジーモデル
c        Model_No.33 = 19            !  両端ピン、中央アナロジーモデル
c
c        Model_No.51 = 51            !  3次元プレテンション動作モデル
c
c        Model_No.50 = 1001          !  DLL 有限要素弾塑性モデル
c
c------------------------------------------------------------------
c
c      ファイバー履歴タイプ
c      nm_type: 1     バイリニア型
c        No.1 = 1                    !  対称バイリニア
c        No.2 = 2                    !  対称トリリニア
c        No.3 = 3                    !  直線コンクリート型
c        No.4 = 4                    !  曲線線コンクリート型
c        No.5 = 5                    !  対称バイリニア型(移動＋等方効果用)
c        No.6 = 6                    !  対称トリリニア型（移動＋等方効果用）
c        No.7 = 7                    !  非対称バイリニア型
c        No.8 = 8                    !  非対称トリリニア型（移動＋等方効果用）
c
c        アナロジーモデル履歴タイプ
c              11     完全弾塑性型
c              12     等方硬化弾塑性型
c              13     移動硬化弾塑性型
c              14     等方硬化＋移動硬化弾塑性型
c
c        マルチスプリング履歴タイプ
c              21     武田モデル
c              22     等方硬化＋移動硬化トリリニア型
c
c------------------------------------------------------------------
c
c      断面モデル
c        No.1                      !  円管
c        No.2                      !  角パイプ
c        No.3                      !  H　型
c        No.4                      !  長方形
c        No.5                      !  T　型
c        No.6                      !
```

```
c         No.7                          !
c
c-----------------------------------------------------------------------
c
c     部材モデルの階層構造は、以下のようである。
c     部材ー＞要素モデルー＞断面モデルー＞履歴モデル
c     １．部材は、各部材固有のデータを保持する。
c     ２．要素モデルは、部材の内部を構成するデータを保持する。
c     ３．断面モデルは、部材の断面形状を等のデータを保持する。
c     ４．履歴モデルは、各要素の弾塑性状態を制御するデータを保持する。
c
c     部材モデルの組み込み手順
c     １．Member_s と Element_s 構造体を設定する。
c     ２．弾塑性解析を実行するために必要となる領域（構造体）を設定する。
c     ３．Cal_stiff_linear に線形剛性を組み込む
c     ４．Cal_stress に応力計算を組み込む
c     ５．Check_stress に弾塑性チェックを組み込む
c     ６．Get_nonlinear_stiff に非線形剛性を組み込む
c
c      解析種別  N_analysis :
c           7: 線形解析
c           8: 幾何学的非線形解析
c           9: 弾塑性解析
c          10: 幾何学的＋弾塑性解析
c           6: 線形座屈解析
c
c      部材別解析種別 : Member(i).nm_analysis
c          -1: 線形解析
c           0: 全体解析種別に従う
c           1: 部材座標系 x 方向弾性
c           2: 部材座標系 y 方向弾性
c           3: 部材座標系 z 方向弾性
c
c-----------------------------------------------------------------------
c
c submain_dynamic_a  引数
c               iend_code       :計算終了コード 0=継続  1:終了
c               icontrol        :計算コード     0:予備計算  1:動的解析 99:終了処理
c               ierr_dat        :エラーコード
c               T               :計算時刻
c               dt              :解析増分時間
c               n_step          :今回の解析ステップ数
c               ns_step         :解析開始ステップ（最初はゼロセットが必要）
c               n_iterate       :反復回数
c-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C     ●   SUBROUTINE /submain_dynamic_S
C      ------------------------------
C          Parallel version for Slave
C      ------------------------------
C     ●   動的解析スレーブ用主プログラム（反復解法）Ver.2.10
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
```

```
c-----------------------------------------------------------------------
      subroutine submain_dynamic_S(i_calnum,iend_code,icontrol,ierr_dat)
c-----------------------------------------------------------------------
c-------------------------------------------------★並列処理に必要な宣言
      use MPI_DEFINE                                                  ! 1
      implicit real*8(A-H,O-Z)
c-------------------------------------------------★計算結果のダンプ出力ファイル番号
      parameter(damp_out = 76)
      character*20 :: strDampFile           ! DOUTPUT_S○のファイル名
c-------------------------------------------------★構造体の定義ヘッダーファイル
      include "..¥..¥sf3st¥submain.h"
      include "..¥..¥sf3st¥submainx.h"
c------- ★★解析制御情報をマスターから取得
c-------------------------------------------------★並列処理用予備計算（処理）（前）
c-------------------------------------------------★MPI 上での自己の状態を得る
c------- ★★
c     自分のランクを取得
      call MPI_Comm_rank(MPI_COMM_WORLD, n_myRank, nErr)              ! 2
c     プロセスの総数を取得
      call MPI_Comm_size(MPI_COMM_WORLD, n_proc, nErr)                ! 3
      write(*, '(A, i3, A, i3)'), "myRank=", n_myRank, "n_proc=", n_proc
c-------------------------------------------------★Doutput の出力先変更
      write(strDampFile, '(A10, i3, A5)') 'DOUTPUT_S', n_myRank, '.txt'
      open(damp_out, file = strDampFile)
c------- ★★担当部材の設定
      call GetRange(1, n_myRank, n_member1)  ! 担当部材の先頭番号     ! 4
      call GetRange(2, n_myRank, n_member2)  ! 担当部材の最終番号
      max_member=n_member2-n_member1+1       ! 担当部材の最大数
      ALLOCATE (pa_work_force(33,max_member))                         ! 5
      ALLOCATE (ff_data(400),if_data(400),iif_dt(2,4))                ! 6
      do i=1, 400
      ff_data(i) = i
      if_data(i) = i
      end do
      write(*,'(a,i5,a,i5,a,i5)')'max_member=',max_member,
     +' n_member1=',n_member1,' n_member2=',n_member2
c      担当部材をモニターより取得
c-------------------------------------------------★モニターからのコントロール情報を取得(ok)
      do i=1,10
      M_alloc(i)=0                    ! 動的配列の確保をチェックする配列をゼロセット
      enddo
      ihan = 0
      ierr = 0
      NFILE=100
      ierr_dat =0
c------- ★★解析制御情報をマスターから取得
      write(*, *) "enter recv_ctlset"
c------- ★★解析制御情報をマスターから取得(vpp)
      call recv_ctlset(Parameter_C,ff_data,if_data,iif_dt,N_analysis,  ! 7
     *                 MPP_Ana_Group,ierr_dat)
c-----------------------------------★制御情報の取得に失敗した場合はここで戻る
      if(ierr_dat .ne. 0 ) then
      DEALLOCATE (ff_data,if_data,iif_dt)
      goto 9997
```

```
      endif
c-----------------------------------------------------★動的解析ダイアログその１のデータをセット(ok)
      is=iif_dt(1,1)                        !  ff_data() 先頭番地
      js=iif_dt(2,1)                        !  if_data() 先頭番地
      call dyctl1_set(ierr,NINDIT,GINDIS,F1SEC,fs_st,fl_st,ifp_st,IST,          ! 8
     *              JIKUZERO,G_JIKUZERO_ALPH,ff_data(is),if_data(js))
c-----------------------------------------------------★動的解析ダイアログその２のデータをセット(ok)
      is=iif_dt(1,2)
      js=iif_dt(2,2)
      call dyctl2_set(ierr,NSTEP,F2SEC,DELT,IGRA,IBETA,BETA,GUMMA,XGAL,          ! 9
     *            NNTIME,EPSDSP,load_memb_mass,dt_M_filter,IT_ANALYS,
     +               ff_data(is),if_data(js))
c-----------------------------------------------------★解析結果の出力パラメータをセット(ok)
      is=iif_dt(1,3)
      js=iif_dt(2,3)
      call doutcl_set(ierr,IWSTP,SOUTSC,DMAXCK,No_section,                       ! 10
     +               ff_data(is),if_data(js))
c-----------------------------------------------------★減衰ダイアログのデータをセット(ok)
      is=iif_dt(1,4)
      js=iif_dt(2,4)
      call damctl_set(ierr,NREAD,ITYDP,NDMP,NDMP2,NHH,HH,QHH,                    ! 11
     +               ff_data(is),if_data(js))

      DEALLOCATE (ff_data,if_data,iif_dt)                                        ! 12
c-------------------------------------------------------------------------
c
c     ★         制御情報を構造体にセット
c
c-------------------------------------------------------------------------
      call Set_control()        !ok (in_disp,in_stres  未定義)
      call Set_model_type()
      call Set_newmark()
      call Set_dynamic_load()
      ・
      ・
c-------------------------------------------------------------------------
c
c     ★         構造・荷重データをマスターから取得する
c
c-------------------------------------------------------------------------
c-----------------------------------------------------★基本構造データをマスターから取得(vpp)
c--------- ★★マスターから構造データを取得
c     バッファデータ仕様
c     buf()     １：座標 3
c               ２：局所座標 3
c               ３：要素 17
c               ４：部材 4
c     ibuf()    １：境界拘束条件 7
c               ２：要素 6
c               ３：部材  14
c
      id_buf=Parameter_C.n_point*6+Parameter_C.n_element*17+                     ! 13
     *       Parameter_C.n_member*4
       jd_buf=Parameter_C.n_point*7++Parameter_C.n_element*6+
```

```
     *       Parameter_C.n_member*14                    ! 全部材数用意する
      ALLOCATE (buf(id_buf+10),ibuf(jd_buf+10))                                  ! 14
       write(*, *) ' allocate ',id_buf, jd_buf
c------- ★★構造データ受信
      call recv_structure()                                                      ! 15
      write(*, *) ' recev_structure ok'
      call set_structure(),                                                      ! 16
      DEALLOCATE (buf,ibuf)                                                      ! 17
      Parameter_C.n_member = max_member     !スレーブ側で計算する担当部材数に変換  ! 18
       .
       .
c-------------------------------------------------★初期不整データをマスターから取得(ok)
c------- ★★マスターから初期不整データを取得
      if(Control.init_imperfection .ne. 0) then
      n_inperfection = 4*Parameter_C.n_point +1      !    初期不整の節点が不明のため節点使用
      ALLOCATE (buf(n_inperfection+10))                                          ! 19
      call recv_imperfection()                                                   ! 20
c      write(*,*) ' recv_imperfection out',Element(1).element_type
      DEALLOCATE (buf)                                                           ! 21
c-------------------------------★情報の取得に失敗した場合はここで戻る
c      if(ierr .ne. 0 ) goto 9997
       endif
c-------------------------------------------------★システム内要素データをマスターから取得
c     ★         ファイバーデータ
c-------------------------------------★ファイバーデータの予備データをマスターから取得(ok)
      n_element=Model_type.n_e_model(11)+Model_type.n_e_model(12)+
     *          Model_type.n_e_model(15)+
     *          Model_type.n_e_model(14)+Model_type.n_e_model(13)+
     *          Model_type.n_e_model(16)+Model_type.n_e_model(17)+
     *          Model_type.n_e_model(18)++Model_type.n_e_model(19)
       if(n_element.ne.0) then
c------- ★★マスターからファイバー用制御データを取得
c      write(*,*) ' recv_Fiber_P 1',n_element,Element(1).element_type
      call recv_Fiber_P()                                                        ! 22
c------- ★★スレーブにファイバー用バッファをゼロセット
      ALLOCATE (buf(id_buf+10),ibuf(jd_buf+10))                                  ! 23
      call recv_Fiber(0,)                                                        ! 24
c-------------------------------★情報の取得に失敗した場合はここで戻る
       .
       .
c-----------------------------------------★ファイバーモデルデータをマスターから取得
c------- ★★マスターからファイバー用データを取得
      call recv_Fiber(1,)                                                        ! 25
      DEALLOCATE (buf,ibuf)
c-----------------------------------------★修正 RO モデル領域をマスターから取得
      n=Model_type.n_m_ro_model
       if(n.ne.0) then
c------- ★★マスターから修正 RO モデル用データを取得
      call recv_RO_data()                                                        ! 26
      ALLOCATE (buf(id_buf+10))                                                  ! 27
      call recv_RO_data()                                                        ! 28
      DEALLOCATE (buf)                                                           ! 29
c-------------------------------------------------★レーリー減衰をマスターから取得(ok)
c------- ★★マスターから減衰データを取得
```

```
      call recv_damp()                                                    ! 30
c-------------------------------★情報の取得に失敗した場合はここで戻る
      if(ierr .ne. 0 ) goto 9997
c-------------------------------------------------------------------------
c
c     ★          動的解析のための予備計算を行う
c
c-------------------------------------------------------------------------
c-----------------------------------------------------★部材長さ計算(ok)
      call Cal_member_length()
c-----------------------------------------------------★モデルの初期設定
      call Set_initial_data()
c-----------------------------------------------------★座標変換行列計算
      call Get_rotate_all()
c-----------------------------------------------------★局所座標変換行列計算
      call Get_rot_local()
c-----------------------------------------------------★釣合座標系標変換行列計算
      call Get_rotate()
c-----------------------------------------------------★節点拘束表の作成
c                                                   未知数等をセット
      call Set_restraint_point()
c-----------------------------------------------------★部材両端の拘束表作成
c-------- ★★節点拘束表の変更（スレーブ）
      call Reset_ij_table()                                               ! 31
c-------------------------------------------------------------------------
      .
      .
      .
c-----------------------------------------★部材両端中央応力、力のゼロセット(ok)
      call  Set_pointforce_zero()
c-----------------------------------------★部材応力のゼロセット(ok)
      call  Set_stress_zero()
c-----------------------------------------------------★MSS モデルの初期設定
      n=Model_type.n_m_model(5)
      if(n.ne.0) then
      call Cal_MSS_dat()
      endif
c-----------------------------------------------------★平面問題における部材の拘束方向チェック(ok)
      call Check_R_direction()
c-----------------------------------------------------★部材の線形剛性計算(ok)
      call Cal_stiff_linear()
c-----------------------------------------------------★剛性の釣合座標系への変換(ok)
      call Rotate_stiffness()
c-----------------------------------------------------★部材の減衰行列計算(ok)
      if(Parameter_C.nc_member .ne. 0) then
      call Cal_damp_linear()
c-----------------------------------------------------★部材減衰行列の釣合座標系への変換(ok)
      call Rotate_damp()
      end if
c-----------------------------------------------------★接線剛性のコピー(ok)
      call Initset_nonlin_stiff()
c-----------------------------------------------------★ベクトルのゼロセット
c-----------------------------------------★増分前の変位、速度、加速度(ok)
      call Set_zero_v()
c-----------------------------------------★最大、最小応力等のゼロセット(ok)
```

```
      call Clear_max_stress()
c-----------------------------------------------------------------------
c
c     ★         静的解析結果等を取り込む
c               （初期変位、初期応力を入力）
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     ★         解析ステップをセットする
c
c-----------------------------------------------------------------------
      ns_step=1
      n_step=10
      d_max_v=0.
      id_max_v=0
      stimes=secnds(0.0)                        ! 解析時間を計る
c-------------------------------------------------★解析手法のセット
      N_implicit_method = 1  !陰解法は１回で反復法に戻る
      Iteration_method  = 1  !最初は反復法を使用
c-----------------------------------------------------------------------

c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     ★         動的解析開始
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
      i_print = 0
 9990 continue                                                          ! 32
      n_iterate = 0                          !  反復回数ゼロセット
      n_member=Parameter_C.n_member
      n_point =Parameter_C.n_point
      n_unknown=Parameter_C.n_unknown
      n_local_coord=Parameter_C.n_local_coord
      write(*,'(a,3i4)') ' Dynamic analysis start :',
     *                ns_step,n_step,Dynamic_load.load_dynamic(1)
      do 9999 istep=ns_step,Newmark_P.nn_step          ! 最後までループを抜けない 33
c------- ★★解析開始コードを受信(mpp)
c           スレーブ制御コード 0:  解析開始　正常終了の場合は最後は制御コードなし
c                              1:  発散・途中終了
c                              2:  エラーは発生、途中終了
c                              3:  未収束
c            call synchronous_process( MPP_Ana_Group)
             call recv_ite(ite_end)                                     ! 34
             if(ite_end.ne.0)go to 9998           ! 解析終了していればループから抜ける
      T=Newmark_P.dt*istep
      i_print=mod(istep,Control.interval_out)
      stimes =secnds(stimes)
       write(*,'(/a,i6,a,f10.3,a,f12.4,a,i4,a,f12.4)')
     *     ' Step No,:',istep,
     *     '     Time :',T,
     *     'sec.     Max. Disp. :',d_max_v,
```

Ver.1.11 では、この制御コード受信用サブルーチンを取り除く

```
      *     '     Node Number : ',id_max_v,
      *     ' times (s):',stimes
        write(damp_out,'(/a,i6,a,f10.3,a,f12.4,a,i4,a,f12.4)')
      *' Step No,:',istep,
      *     '     Time : ',T,
      *     'sec.     Max. Disp. : ',d_max_v,
      *     '     Node Number : ',id_max_v,
      *     ' times (s):',stimes
        stimes=secnds(0.0)
c-----------------------------------------------------------------------
c
c     ★         動的解析の開始
c
c-----------------------------------------------------------------------
      if(Control.type_analysis .ne. 7) goto 9981
c-----------------------------------------------------------------------
c
c     ★         線形解析
c
c-----------------------------------------------------------------------
c-------------------------------------------------------★右辺の定数ベクトルゼロセット(ok)
      call Clear_vec()
c-------------------------------------------------------★線形減衰項計算(ok)
c-----------------------------------------------★整合質量(ok)
      call Add_damp2_ld()
c-----------------------------------------------★部材減衰(ok)
       call Add_damp3_ld()
c-------------------------------------------------------★線形剛性項計算(ok)
c                                                      レーリー減衰も含む
      call Add_stiff1_ld()
c-------------------------------------------------------★Δｔ秒後の変位と速度を予測(ok)
      nx =0
      call Estimate_disp_vel()
c-------------------------------------------------------★Maxwell 型モデルの計算(ok)
      call Add_fdd_ld()
c-------------------------------------------------------★右辺項を送る (vpp)
c-------- ★★マスターに右辺項を送信
c      write(76, '(a, i5)') 'Send_ld_repeat:istep = ', istep
      call send_ld_repeat()
c-------------------------------------------------------★サーバーで計算した加速度を受け取る (vpp)
c-------- ★★マスターから計算結果加速度を取得
       call recv_result_acc()
c-------------------------------------------------------★加速度よりβ法に基づき変位と速度を計算(ok)
      call Cal_disp_vel()
       goto 9980
c-----------------------------------------------------------------------
c
c     ★         非線形解析
c
c-----------------------------------------------------------------------
 9981 continue
c-----------------------------------------------------------------------
c
c     ★         動的解析（反復解法）
```

```
c
c-------------------------------------------------------------------------
c-----------------------------------------------------★右辺の定数ベクトルゼロセット(ok)
      call Clear_vec()                                                ! 35
c-----------------------------------------------------★部材節点力のセット(ok)
       call Get_pointforce_ld()
c-----------------------------------------------------★線形減衰項計算(ok)
c-------------------------------------------★整合質量(ok)
      call Add_damp2_ld()
c-------------------------------------------★部材減衰(ok)
      call Add_damp3_ld()
c-----------------------------------------------------★線形剛性項計算(ok)
c                                                      レーリー減衰も含む
      call Add_stiff1_ld()
c-----------------------------------------------------★Δｔ秒後の変位と速度を予測(ok)
C      n_err_roop = 0
 9991 continue
      nx =0
      call Estimate_disp_vel()
c-------------------------------------------------------------------------
c
c    ★          反復計算開始
c
c-------------------------------------------------------------------------
      n_roop=Newmark_P.max_repeat
      do iroop=1,n_roop
c-----------------------------------------------------★反復に関連する右辺ベクトルのゼロセット(ok)
       call Clear_vec()                                               ! 36
c-----------------------------------------------------★線形剛性に関するベクトル(vpp)
       call Add_stiff2_ld()
c-----------------------------------------------------★接線剛性に関する増分ベクトル(vpp)
       call Add_tan_stiff_ld()
c-----------------------------------------------------★Maxwell 型モデルの計算(vpp)
       call Add_fdd_ld()
c-----------------------------------------------------★右辺２項の和を取る(ok)
       call add_vec()
c-----------------------------------------------------★右辺項を送る （vpp)
c-------- ★★マスターに右辺項を送信
       call send_ld_repeat()                                          ! 37
c-----------------------------------------------------★サーバーで計算した加速度を受け取る （vpp)
c-------- ★★マスターから計算結果加速度を取得
       call recv_result_acc()                                         ! 38
c-----------------------------------------------------★β法に基づき加速度より変位と速度を計算(ok)
       call Cal_disp_vel()
c            スレーブ制御コード 0:  解析開始　正常終了の場合は最後は制御コードなし
c                               1:  発散・途中終了
c                               2:  エラー発生、途中終了
c                               3:  未収束
c 同期
c             call synchronous_process( MPP_Ana_Group)
              call recv_ite(ite_end)                                  ! 39
              if(ite_end.eq.0)go to 9980                  ! 収束していればループから抜ける
              if(ite_end.eq.2)go to 9987
c-----------------------------------------------------★次の増分値を予測(ok)
```

Ver.1.1 では、加速度と同時に制御コードを受け取る

Ver.1.11 では、この制御コード受信用サブルーチンを取り除く

```
      nx=iroop
      call Estimate_disp_vel()
      end do
c-----------------------------------------------------------------------
c
c     ★        収束しなかった時の後処理
c
c-----------------------------------------------------------------------
9987 continue
      write(76,*) '  収束エラー発生、処理中止'
      goto 9997                                                    ! 40
c-----------------------------------------------------------------------
c
c     ★        計算終了・後処理開始
c
c-----------------------------------------------------------------------
 9980 continue
c-------------------------------------------------★解析結果・増分変位をセット(ok)
      call Set_ddisp()                                             ! 41
c-------- ★★  Parallel Code -Start (Stress)
c-------------------------------------------------★部材応力を計算(vpp)
      call Cal_stress()
c-----------------------------------------------------------------------
c
c     ★        部材の弾塑性状態をチェック
c
c-----------------------------------------------------------------------
c-------------------------------------------------★部材塑性をチェック
c-------------------------------------------------★部材両端の節点力計算 (ok)
c-------------------------------------------------★ファイバー応力セット(vpp)
      call Check_stress()
c-------------------------------------------------★部材の節点力と部材応力をマスターに送る(vpp)
c-------- ★★マスターに部材応力を送信（マスターがファイル出力する時のみ）
       if(i_print.eq.0) then
      call send_force_vpp()                                        ! 42
      endif
c-------------------------------------------------★Maxwell 型モデルの非線形性チェック(ok)
      call Check_Maxwell_stress()
c-------------------------------------------------★応力等の最大値セット
      call Get_max_stress()
c-------------------------------------------------★解析結果・変位、速度、加速度をセット(ok)
      call Set_disp_vel_acc()                    ! past_dacc_point に過去の加速度をセットする
c-------------------------------------------------★終了か？（ステップと最大値チェック）
      if(istep .ge. Newmark_P.nn_step  ) goto 9998                 ! 43
c-------------------------------------------------★接線剛性の計算(ok)
      call Get_nonlinear_stiff()
9999 continue

c-----------------------------------------------------------------------
c
c     ★        応答解析終了処理
c
c-----------------------------------------------------------------------
 9998 continue
```

```
c―――――― ★★　最大応力等を送信
      call send_max_stress()                                          ! 44

 9997 continue
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
c
c     ★          動的配列の解放
c
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
c――――――――――――★          配列の動的領域を解放する（その１）　ok
         .
         .
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
c
c     ★          並列処理で使用した動的配列の解放
c
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
      DEALLOCATE (pa_work_force)                                      ! 45
c――――――――――――――――――――――――――――――――――――★並列処理通信グループの解法

c――――――――――――――――――――――――――――――――――――★計算終了コードセット
      iend_code =1
      close (damp_out)
      return
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
c
c     ★          解析終了
c
c―――――――――――――――――――――――――――――――――――――――――――――――――――――
      end
```

プログラムの内容をコードの右端のコメント番号に従って説明する。この主サブルーチンの基本的な解析の流れに関する解説は、マニュアル：動的解析編を参照されたい。ここでは、並列処理に関連する部分について説明する。

1. 変数のバイト数など、MPI 準拠に基づいた定数宣言用ファイル MPI_DEFINE を、use 文を用いてインクルードする。
2. 分散並列システムより、自分のランク（あるいはプロセス ID） をシステムより取得する。
3. 並列システムを構成するプロセスの総数をシステムより取得する。
4. このスレーブの担当部材の先頭部材番号と最終部材番号を取得する。また、担当部材数をセットする。
5. 担当部材の応力に関するマスター転送用バッファ領域を動的確保する。
6. 解析制御用パラメータをパックしたデータを、マスターより受け取

るためのバッファ領域を動的に確保する。
7. マスターからの解析制御用パラメータをバッファ領域に受信する。
8. バッファ領域から動的解析ダイアログその 1 の該当する変数にデータを取り出す。
9. バッファ領域から動的解析ダイアログその 2 の該当する変数にデータを取り出す。
10. バッファ領域から解析結果の出力パラメータの該当する変数にデータを取り出す。
11. バッファ領域から減衰ダイアログの該当する変数にデータを取り出す。
12. バッファとして使用した動的領域を解放する。その後、4 つのサブルーチンを用いて、制御パラメータを構造体にセットする。
13. 構造データを受信するためのバッファ領域の大きさを計算する。ここで使用する節点数や部材数は、既に制御パラメータ受信の中で得られている。
14. 構造データ受信用バッファ領域を動的に確保する。
15. 構造データを受信する。
16. この受信したバッファ領域の中の構造データを用いて、構造体にセットする。この時、部材に関する構造体 Member には、このスレーブが分担する部材に関するデータのみ保存される。このスレーブで解析する部材数もこの分担した部材数となる。
17. バッファ領域を解放する。
18. 分担した部材数である解析部材数を構造体にセットする。
19. 初期不整が存在する場合は、マスター側からデータを受信するためのバッファが必要となる。ここでは、その大きさが分からないため、節点数を用いて、バッファ領域を動的に確保する。
20. 初期不整データをマスターから受信し、節点座標に加える操作を行う。
21. バッファ領域を解放する。
22. ファイバー断面を用いる場合、ファイバーデータを受信するための個数が、先にマスターから送られてくる。ここで、そのデータを受信する。
23. 上記の個数データを用いて、バッファ領域を動的に確保する。
24. ファイバーデータを受信する。このサブルーチンは 2 回コールされ、データ受信と共に、構造体へのデータセットや各種の情報処理を行う。

25.2 回目のサブルーチンコールが行われ、構造体にデータがセットされ、情報処理が行われる。その後、バッファ領域を解放する。
26.修正 RO モデルを使用する場合、マスター側よりこのモデルに関するデータが送信される。最初にデータの送信個数が送信されるので、このサブルーチンでこの個数データを受信する。
27.個数データを用いて、バッファ領域を動的に確保する。
28. 修正 RO モデル用のデータを受信し、該当する構造体にデータをセットする。
29.動的バッファ領域を解放する。
30.レーリー減衰用データを受信する。送信個数が決まっているので、一回のサブルーチンコールでデータを受信することができる。また、受信したデータを該当する構造体にセットする。その後の処理は、単一 CPU の処理内容と同じであり、同一のサブルーチンを使用する。ただし、部材に関する処理は、各スレーブが分担する部材についてのみ行われる。
31.上のサブルーチンで節点に関する拘束表から未知数番号表を作成するが、ここでは、その未知数番号表を作り直す。具体的には、分担した部材に接続する節点にのみ未知番号を付け直す。さらに、部材に関する両端の未知番号表を作成する。この表を利用することで、マスターから送信されるパックした状態の加速度ベクトルを直接にアクセスすることが可能となる。これ以後の予備計算におけるサブルーチンは、単一 CPU に対する処理と同じとなる。ただし、出力ファイルのオープン処理は省かれている。
32.マスター側では、予備計算が終了すると動的ソルバーから一旦抜け、動的解析システムに制御が移され、その後マルチスレッド処理によって、ひとつのスレッドが再度動的ソルバーに制御を移すことになる。しかしながら、スレーブ側では、コンソールアプリケーションであるため、予備計算から連続して動的解析ルーチンへと進む。
33.スレーブ側では、動的ソルバーを抜けることなく、最後まで処理を続ける。
34.マスター側からの解析制御用コードを受信する。コードが 0 以外の場合は解析終了であり、後処理へ制御を移す。
35.非線形解析の中で、右辺項を計算する。ここ以降では、分担部材に対応する右辺項を求めるが、最初は増分後の変位、加速度に関連しない項を求める。
36.反復計算ループ内の増分後の変位、加速度に関連する右辺項を計算

Ver.1.11 では、この制御コード受信用サブルーチンを取り除く

する。ここでも、このスレーブが分担する部材について右辺項を求める。反復ループ内の右辺項とループ外の右辺項の和を取る。

37.上で和をとった右辺項をマスター側に送信する。

38.マスター側で方程式を解いて得た加速度ベクトルが、各スレーブに合わせてパックされ、送信される。このパックされた加速度ベクトルを受信する。

39.制御コードを受信する。コードが 0 の場合は収束で、文番号 9980 へ制御を移し、コードが 2 の場合は最大反復ループを超過したため、文番号 9887 へ移動する。コードが 3 の場合は再度反復計算を実行する。

40. 最大反復ループを超過したため、エラー表示を行った後、後処理へ制御を移す。

41.反復計算が終了した後、分担した部材に対し、応力計算、弾塑性チェックを行う。

42.分担した部材応力をマスター側に送信する。ただし、マスター側でファイルに出力するタイミングで送信する。

43.最大ステップ数を越えたかどうかチェックし、全て計算を終えると後処理へ制御を移す。

44.分担した部材の最大応力をマスターへ送信する。

45.並列処理用のワーク動的領域を解放する。

Ver.1.1 では、加速度と同時に制御コードを受け取る

Ver.1.11 では、この制御コード受信用サブルーチンを取り除く